pdp11

# RSTS/E
## System User's Guide

Order No. DEC-11-ORSUB-A-D

digital

# RSTS/E

## System User's Guide

Order No. DEC-11-ORSUB-A-D

digital equipment corporation · maynard. massachusetts

First Printing, December 1976

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

# CONTENTS

# CONTENTS (Cont.)

# CONTENTS (Cont.)

# CONTENTS (Cont.)

## CONTENTS (Cont.)

CONTENTS (Cont.)

## CONTENTS (Cont.)

# CONTENTS (Cont.)

# CONTENTS (Cont.)

# CONTENTS (Cont.)

# PREFACE
# USING THIS DOCUMENT

This document describes the RSTS/E system and its resources; it is divided into five parts, and organized as follows:

PART I, "SYSTEM FUNDAMENTALS AND RESOURCES," introduces the new user to the RSTS/E system. Part I explains fundamentals such as timesharing theory, entering and leaving the system, how data is organized and located, and what Input/Output devices are available.

PART II, "ADAPTING SYSTEM RESOURCES," explains the use of system resource commands, which allocate and manipulate devices and adapt certain system characteristics to the needs of the individual user.

PART III, "BASIC-PLUS SYSTEM COMMANDS, " explains the use of BASIC-PLUS system commands, which enable the user to write, edit, modify, debug, and manipulate BASIC-PLUS programs.

PART IV, "SYSTEM LIBRARY PROGRAMS," describes and explains the use of the RSTS/E system library programs, which perform various functions of general utility. Among these functions are editing text, transferring data, saving data off-line, printing timesharing statistics, setting terminal characteristics, and batch processing. The overview ("Programs at a Glance") at the beginning of Part IV lists these system library programs by function.

PART V, "RSTS/E PERIPHERAL DEVICES," describes some Input/Output devices available to the RSTS/E user, and explains their hardware settings and operations.

The reader's approach to the User's Guide will depend upon his or her experience with computer systems and languages in general, and with RSTS/E and BASIC-PLUS in particular.

The beginning user, who may have little or no computer experience, should first read Parts I and II, in order to gain an overall understanding of the RSTS/E system and its resources. Anyone programming in the BASIC-PLUS language must use the BASIC-PLUS system commands in order to write and modify programs. The newcomer to BASIC-PLUS, therefore, may expect to use Part III concurrently with the *BASIC-PLUS Language Manual,* which describes the BASIC-PLUS language itself.

The intermediate user, who typically may have some BASIC-PLUS programming experience, and may be still learning about the RSTS/E system, would probably employ Parts III and IV largely as references.

The experienced user, who is familiar with both BASIC-PLUS and RSTS/E, would probably refer most often to Part IV, since some of the system programs it describes are new or modified since the previous version of RSTS/E.

All users should find Part IV particularly useful as a reference to the system programs. Also intended for general reference are Parts II, III, and V.

For information on other RSTS/E manuals, see the *RSTS/E Documentation Directory.*

# PART I

# SYSTEM FUNDAMENTALS AND RESOURCES

# CHAPTER 1
# TIMESHARING AND RSTS/E

## 1.1 INTERACTIVE TIMESHARING IN COMPUTER EVOLUTION

The development of interactive timesharing represented a major step in the evolution of modern computers. To understand this concept, one must consider the two terms, "interactive" and "timesharing," and what they mean in the context of computing history. Early computers were designed to run without any human intervention; an operator would "feed" stacks of punched cards into the computer system, then wait – patiently or impatiently – while the job was processed. If there were errors, typographical or otherwise, lurking unseen among the thousands of stacked cards, the programmer would not discover them until the system had completed the entire job. Furthermore, correcting these errors involved first hunting them down through reams of computer listings, then repeating the laborious, time-consuming process of punching the cards. And, since the cards were prepared by human keypunch operators, there usually were some errors on the first "pass" or submission of cards to the system.

Thus, a critical drawback of early computers was that they were not interactive. While they were processing jobs, they did not allow users to communicate with them in order to discover errors, check results, or modify programs.

Another disadvantage of many early computers was their inability to process more than one job at a time. This limitation, combined with an inability to interact with the user, made the early computer unattractively slow and expensive for many widely desired applications. Thus, the computer was used mainly for highly repetitive problems that did not require rapid "turn-around" – that is, fast return of processed results.

Timesharing, however, made the computer more widely available. The image of the electronic oracle whose slow and mysterious ponderings were presided over by a privileged cult of programmers and mathematicians had begun to fade.

Interactive timesharing systems are changing this rather restrictive image to a "friendlier" one. For not only can users communicate with such a system while it processes their data, but they can share its resources concurrently. A timesharing system assigns each user a quantum or "slice" of its processing time, and allocates service among programs until each program has been completed. The rapidity of this scheduling has an important – and generally gratifying – psychological effect: each user enjoys the pleasant illusion that the system "belongs" to him or to her alone.

## 1.2 HOW TIMESHARING WORKS

To understand how a timesharing system works, one may imagine a typical RSTS/E system in operation, and currently serving 10 users, each seated at a terminal and communicating with the system. Of these 10 users, 6 are running their own programs: 4 in the BASIC-PLUS language, 1 in the COBOL language, and 1 in the FORTRAN language. Of the remaining 4 users, none is working directly with a computer language; instead, these 4 are running system programs that perform special tasks. One user is running EDIT, a program that enables him to write, modify, and correct text. Another is running the DIRECT program, which prints at the terminal a list or directory of information she has previously stored in the system. The remaining two users are running PIP, the Peripheral Interchange Program, which transfers information from one Input/Output device to another: for example, from a disk to a line printer, which outputs text a line at a time. Figure 1-1 illustrates the system's current usage.

Not only must the RSTS/E system perform different functions for each of these 10 users, but it must also keep these functions in separate sets (one to a user), allocate a "slice" of time to each set, and schedule these "slices" so that each user receives one at short intervals.

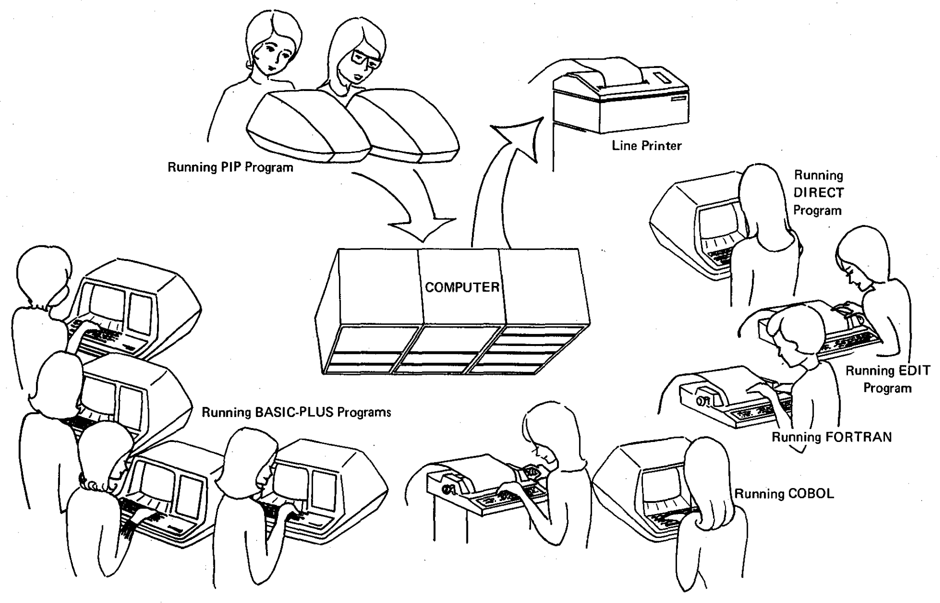


Figure 1-1 Users Sharing Time on the RSTS/E System

How does RSTS/E arrange to schedule and process each user's information efficiently and without confusion? First of all, RSTS/E "sees" each user's set of operations – commands, input of data, etc. – as a separate job, which exists for as long as that user is "on" the system. To RSTS/E, therefore, a job is a sequential string of associated operations: the unit of identification by which it "knows" one user from the others. These jobs all exist on a computer disk in an area known as the swapping space. When a job in memory – that is, a user – asks the system to run one of its own system programs, such as PIP, the request goes to two areas in the computer's memory: first to the BASIC-PLUS run-time system, then to the executive. Figure 1-2 shows several concurrently running jobs sending requests along this route. Job number 1 happens to be sending the request that the system run the program called PIP.

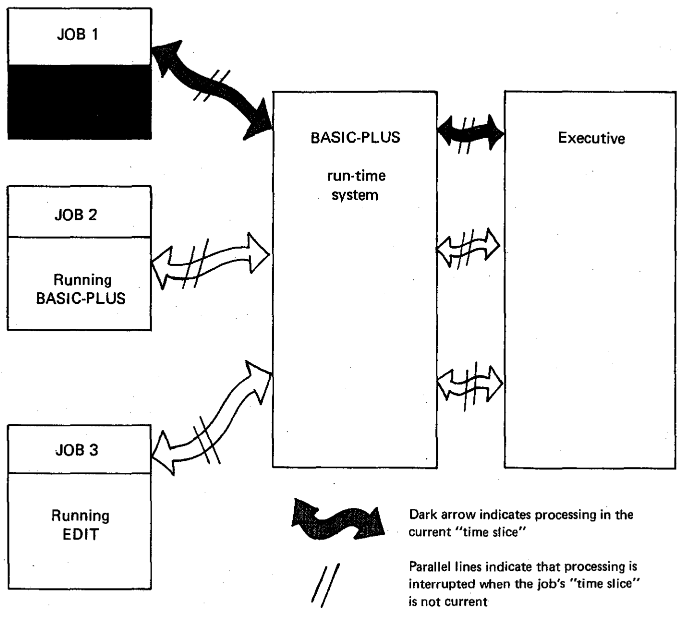

JOB 1
JOB 2
Running BASIC-PLUS
JOB 3
Running EDIT
BASIC-PLUS
run-time system
Executive
Dark arrow indicates processing in the current "time slice"
Parallel lines indicate that processing is interrupted when the job's "time slice" is not current

Figure 1-2 The Timesharing Process

When a job makes a request, that request is first interpreted by the BASIC-PLUS run-time system, hereafter referred to as the BASIC-PLUS system. Depending on the nature of the request, the BASIC-PLUS system fulfills it, or passes it on to the executive. The executive, as its name denotes, executes or carries out the request. It then returns control to the BASIC-PLUS system.

When job number 1, for example, asks for PIP, the BASIC-PLUS system "fields" the request: interprets it and passes it to the executive. The executive, in turn, acts on the request by attempting to "open" the prescribed set of stored data – or the file – that is known as the PIP program. If all goes well and the executive succeeds in opening that file, it then gives the BASIC-PLUS system access to PIP, and also returns control to that system. Now, the BASIC-PLUS system acts as interpreter of program statements that are taken from the PIP program.

The situation just described, in which a user runs a system program like PIP, affords a further example of the typically busy interaction between the BASIC-PLUS system and the executive. For there are many commands that the user may give to the PIP program – commands that cause data to be printed out, summarized, merged, transferred from one storage device to another, even completely erased from the system. In the course of a single job, the user may issue all of these commands to PIP. Each command would follow the route described: to the BASIC-PLUS system for interpretation, then to the executive for carrying out. The executive, moreover, in addition to carrying out the commands of other current users, would be performing its other vital "housekeeping" functions: allocating and scheduling the jobs, thereby making sure that each user, at regular intervals, receives a rightful "slice" of the timesharing pie.

Essential to the executive's efficiency is not only its speed, but also its ability to keep each job separate and distinct, isolated from all other current jobs. Thus, it is able to "swap" jobs in and out of the computer's memory as their time slices begin and end. These two facilities – speed and job separation – preserve each user's illusion of "owning" the system exclusively. For the user receives responses that are not only fast, but uncluttered by irrelevant information pertaining to other jobs. A brief example shows how confusing a system's response would be were it not for job separation and swapping. What would happen, for instance, if two user jobs were printing, at the line printer, two very different texts: a work of fiction and a recipe? Were it not for job separation, the line printer output might look like this:

```
MURRAY THE RABBIT LIVED IN A NICE            (USER 1)
RABBIT STEW: BEGIN WITH A PLUMP,             (USER 2)
LITTLE TOWN WHERE EVERYONE LOVED             (USER 1)
FRESHLY KILLED RABBIT. FIRST, SKIN IT BY     (USER 2)
DANCING AND FROLICKING ON THE GREEN.         (USER 1)
```

Clearly, these two users have different things in mind, and both would be confused, possibly upset, by the mixed output. But the executive, by separation, assures that no two jobs can use the line printer at the same time.

Thus, the timesharing process, seen from inside the system, can be quite complex, particularly when one realizes that a RSTS/E system, with enough hardware, can support up to 63 concurrent users. The reader of this manual, however, will probably not be immediately concerned with the internal complexities of job swapping and scheduling. Knowledge of such intricacies, while interesting and of increasing value as one becomes more experienced with computers, is not necessary in order to use any of the RSTS/E resources described here in the User's Guide, or to use the BASIC-PLUS language.

## 1.3 THE USER'S VIEW OF TIMESHARING

Of immediate interest to the reader is an orientation to RSTS/E timesharing: a general concept of how timesharing appears to the user who is "on" the system and working at a terminal. Assume that this user, in the course of his job, performs a typical sequence of operations: writing his own program, running it, requesting one of the system's programs, giving that program a command, and finally asking the system to reprint – at his terminal the program he wrote in the first operation. These steps are listed below, along with brief descriptions of how the user perceives the timesharing system during each step.

1. The user is writing his own program in the BASIC-PLUS language.
   The user sees himself in communication with the BASIC-PLUS language; he types a line number, a statement, and the RETURN key, repeating this procedure until the program is complete. If he makes a syntactical error, BASIC-PLUS communicates in turn with him, printing an error message – a brief description of the mistake.
2. The user, having written the BASIC-PLUS program, asks that it be run by typing the RUN command.

   READY: BASIC-PLUS command level

   After thus running the program, the user sees himself in communication with the BASIC-PLUS (run-time) system, because the word READY appears at the terminal. READY is the BASIC-PLUS system's unique prompt: its way of telling the user it is READY to accept commands. This prompt, of course, also lets the user know he is currently in communication with the BASIC-PLUS system. As long as the user keeps encountering this prompt after execution of commands, he is said to be at BASIC-PLUS command level.
3. The user responds to the READY prompt by typing SAVE to store his newly created program on the disk.
   After the program has thus been saved, the READY prompt appears once more. The user again sees himself in communication with the BASIC-PLUS system – that is, at BASIC-PLUS command level.
4. The user, responding again to the READY prompt, types the command RUN $PIP, thus gaining access to the system's program PIP, whose functions are data transfer, input, and output.
   Now, the user sees himself in direct communication with the PIP program – in other words, at program level.

   # prompt: program level

   The user perceives himself at this level because PIP immediately identifies itself by printing its name (and other information) at the terminal, because the prompt changes from the word READY to a number sign (#). This number sign is PIP's own prompt. Like READY, it tells the user that a command will be accepted. It also tells the user, of course, that he is no longer at BASIC-PLUS command level, but is now at PIP program level.
5. To PIP's # prompt, the user responds by typing a PIP command that causes a set of his stored data (a file) to be printed out at the line printer.
   To the user, it appears that PIP fulfills this request and awaits another command, because once again the # prompt appears at the terminal.
6. Since the user has no additional commands for the PIP program, he dismisses it by pressing a special combination of keys ("CONTROL" and "Z"), and once again sees the READY prompt. Thus, he is back at BASIC-PLUS command level, and is in communication with the BASIC-PLUS system.

   READY again: back to BASIC-PLUS command level
7. Responding to the READY prompt, the user types a series of commands that access his BASIC-PLUS program and print it at the terminal.

The reader will note, from this typical sequence, that the RSTS/E user is unaware of the executive's role in timesharing operations. To be sure, it has been quite busily involved in this sequence – receiving and acting upon information from the BASIC-PLUS system, providing this user and others with their "time slices" at regular intervals, and so forth. But as far as the user is concerned, the dominant, supervisory "intelligence" on which he has depended for allocation of resources (such as the BASIC-PLUS language and system programs) is the BASIC-PLUS system. Although the executive may be the power "behind the scenes," so to speak, the BASIC-PLUS system is the computer's "front office", or the user representative which entertains requests or commands and ensures that they are carried out. Indeed, to most users, RSTS/E is BASIC-PLUS: the system and the language.

Certainly, other languages – COBOL and FORTRAN – are available on many RSTS/E systems, and users of these languages will be communicating with other run-time systems, described in the manuals dealing with those languages. Users concerned with COBOL and FORTRAN should refer to the appropriate manuals for such information, which is beyond the scope of the RSTS/E User's Guide.

# CHAPTER 2
# THE SYSTEM FUNDAMENTALS

## 2.1 BECOMING A RSTS/E USER: PROJECT-PROGRAMMER NUMBER AND PASSWORD

In order to become a user, or timesharer on the RSTS/E system, one must be assigned two codes by the system manager: a unique project-programmer number and a password. These enable the user to "log in" – to gain access to the RSTS/E system and its resources, which include I/O (Input and Output) devices, programs of various functions, information stored by the user himself, the BASIC-PLUS language, and optional resources such as COBOL and FORTRAN-IV. Together, the project-programmer number and the password constitute the user's identification to the RSTS/E system, the means by which it recognizes him as an authorized sharer of its time and resources. Each time that a user attempts to log into the system, the system will ask for both these codes. If the user correctly types his assigned number and password, RSTS/E will log him into the system; but if his response is incorrect, it will not – that is, it will deny him access.

A project-programmer number, as typed by the user, looks like this:

```
100,101
```

When printed by the system or in the text of this manual, the project-programmer number is enclosed in square brackets or in parentheses: [100,101] or (100,101). In this code, 100 is the project number, possibly shared by a group of users with a common activity or interest; 101 is the programmer number, held by only one user within the project group. Thus, each user at a RSTS/E site has a unique project-programmer number. These unique numbers are used to identify sets of data (files) according to the users who "own" them, and to provide the basis by which these files are protected from access by certain groups of users. (File protection is discussed later, in Section 2.4.2.) The project-programmer number is also called the account number, because it identifies the user's own reserved area for data storage, or his account.

The user's password is an alphanumeric code assigned him; when he types it in during the login procedure, it is not printed on the paper or displayed on the screen. This secrecy is intended to prevent unauthorized persons from reading a user's password and thus gaining illegal access to the system or to that user's data.

**NOTE**

To preserve the security of the system, a user should memorize his password as soon as it is assigned, rather than write it out.

## 2.2 LOGGING IN: THE HELLO COMMAND

Once the user has been assigned a project-programmer number and password, he may log in at a terminal. First he should make sure that the terminal's LINE-OFF-LOCAL knob (or switch)[1] is set to LINE. This setting establishes a line of communication from terminal to computer; when this communication exists, the terminal is said to be on-line.

Once the terminal is on-line, the user begins the login procedure by typing the command

```
HELLO
```

and then pressing the RETURN key. Thus the user tells RSTS/E that he wishes to join the system. RSTS/E responds to the user's HELLO by printing a system identification line and then a number sign (#), which appears

[1] Alternate types of user terminals may have a different knob or switch designed to put them on-line. See Part V or the appropriate hardware terminal manual.

at the left margin of the paper or screen. This number sign is the system's prompt to the user; it tells him that the system is waiting for him to type his project-programmer number and to press RETURN. Once these actions have been performed, the system responds by again prompting the user, this time by printing:

PASSWORD:

The system then waits for the user to type his password and to press RETURN. Note that as the user types his password, it is not printed at the terminal.

If the codes are acceptable to the system, the user is logged in, and the system prints the daily message at his terminal. This message, written by the system manager, contains information on any changes or additions to the system. Once the user is logged into the system, the terminal becomes the user's console terminal – the terminal that initiated his access to the system.

If the codes entered are incorrect, the system prints

INVALID ENTRY – TRY AGAIN

and the user may attempt once more to log in.

After five unsuccessful attempts to log in, the system prints the message ACCESS DENIED and ends the login procedure.

The entire process of entering the system is shown in the example that follows. Note that although the RETURN key is pressed to enter each line to the system, it does not echo (appear in print) on the terminal paper or screen. It does, however, have a visible effect, which is to perform a carriage return/line feed operation. In this example, as in all others in this manual, the user is employing an LA36 (DECwriter II) terminal. Characters typed by the user are underlined to differentiate them from characters printed by the system.

```
HELLO

RSTS V06B-02 Timesharing  Job 29  KB3  28-Oct-76  01:57 PM
#2,201
Password:


17-Oct-75
          TO ALL USERS--
          RSTS/E TIMESHARING HOURS WILL BE FROM
          9:30 AM TO 7:30 PM TODAY.  FOUR NEW
          TERMINALS ARE AVAILABLE IN THE SYSTEM
          ROOM FOR GENERAL USE.


Ready
```

The READY prompt following the daily message is first printed when the user is successfully logged into the system. READY is preceded by a carriage return/line feed operation and followed by a carriage return and 2 line feeds. It thereafter indicates to the user that he is at command level, a condition of operation in which he may issue any valid RSTS/E command. At command level, for example, the user may type NEW to create a BASIC-PLUS program, or OLD to retrieve one already saved. Or, the user may type any of the other BASIC-PLUS system commands, any of the resource commands, or any of the commands that run RSTS/E system library programs. (These types of commands are described in Parts II, III, and IV of this manual.)

Another way to log into the system is to enter the project-programmer number on the same line with the HELLO command, as shown in the following example. This action causes the system to print the PASSWORD: prompt only. The user's input is underlined.

```
HELLO 200/57
Password:

Ready
```

Note that in the preceding example, the user types a slash (/) rather than a comma to separate the project and programmer numbers. The slash causes no daily message to be printed.

## 2.3 LOGGING OUT: THE BYE COMMAND

When the user wishes to end his timesharing and leave the terminal, he may type the command

```
BYE
```

and then press the RETURN key. Thus the user tells RSTS/E that he wishes to leave the system. RSTS/E responds to the BYE command by printing the prompt

```
CONFIRM:
```

As the CONFIRM: prompt indicates, RSTS/E is waiting at this point for the user to confirm his intention to leave the system. The simplest and most obvious reply is to say "yes"; this the user does by typing Y, then pressing RETURN. In response, RSTS/E will dismiss him from the system – i.e., log him out – printing a statistical report on his storage area and his timesharing session as it does so. This report tells the user how much of his allotted space he has used, and confirms his identity and that of the system. It also informs him about time: how much has elapsed since he logged in, and how much of that time was spent processing his input, and the approximate time of his logout.

On the other hand, the user, since typing BYE, may have changed his mind about leaving the system. Perhaps he has remembered that he intended to modify a BASIC-PLUS program, or proofread a passage of text; for whatever reason, he wishes to stop the logout sequence, and remain at RSTS/E command level. Obviously, he needs to say "No" to the CONFIRM: prompt. This he does by typing N, then pressing RETURN. RSTS/E responds by returning him to command level and printing the READY prompt.

Y and N are two of the five possible replies a user may give to the CONFIRM: prompt. The other replies are: I, which allows individual examination and deletion of files; ?, which requests a printing of CONFIRM: replies; and F, which results in a Fast logout. All five replies are described in the following list:

| **The CONFIRM: reply** | **Means** |
|---|---|
| N | No logout is performed, and the system replies READY. |
| I | Individual file descriptions are printed, each followed by a ?. To delete a file, the user types K and the RETURN key; to retain it, just the RETURN key. |
| ? | A listing of CONFIRM: replies is printed. |

| **The CONFIRM: reply** | **Means** |
|---|---|
| Y | Yes. Logout is performed, if the user has not exceeded his disk quota; if he has, RSTS/E will ask that he delete some file(s). To log out, he must be within quota. |
| F | Fast logout is performed (same effect as Y except that the report is not printed and 3 form feeds are generated). |

**NOTE**

Users unfamiliar with RSTS/E files, mentioned in the descriptions of I and Y, should read Section 2.4, "THE DIRECTORY AND FILES."

For a full description of the logout procedure, including an example of the I response, see the LOGOUT program in Chapter 14.

In the following example, the user logs out by typing Y.

```
BYE
Confirm: Y
Saved all disk files; 204 blocks in use
Job 29 User 2,201 logged off KB3 at 28-Oct-76 01:58 PM
System RSTS V06B-02 Timesharing
Run time was .3 seconds
Elapsed time was 26 seconds
Good afternoon
```

Before leaving the terminal, the user should turn the LINE-OFF-LOCAL knob or switch to OFF. (The LOCAL setting leaves the terminal with power, but disconnected from the system; it then operates as a typewriter.)

## 2.4 THE DIRECTORY AND FILES

Once a user has written information into his account, he is said to have a directory: a list of the files he has created and stored, along with information about them – individually and as a group. Files, on the RSTS/E system, are separate and distinct programs, bodies of data, or empty areas designated for future input.

Perhaps the best way to describe the user's directory is to look at an example. To print the directory at his terminal, the user types the command DIR in response to the system's READY prompt. Many users customarily request their directories immediately after logging in; thus, they see at a glance the current status of their work on the system, and are reminded of the names they have chosen for their files.

In the following example, a user, logged in from an LA36 (DECwriter) terminal, requests his directory, which is then printed out by the system:

```
DIR
 Name .Ext  Size   Prot       Date
AVERAG.BAS     2  < 60>   28-Oct-76   SY:[200,57]
PERCNT.BAS     2  < 60>   28-Oct-76
TERM  .DOC    11  < 60>   28-Oct-76
REPDLO.TXT    43  < 60>   28-Oct-76
```

```
PHONE .DIR     43   < 60>    28-Oct-76
EXPENS.BAS     29   < 60>    28-Oct-76
CHOICE.BAS      2   < 60>    28-Oct-76
SOURCE.DAT     20   < 16>    28-Oct-76
BACKUP.CMD      1   < 60>    28-Oct-76
XX    .TST      1   < 60>    28-Oct-76
VISITS.LST     14   < 40>    28-Oct-76

Total of 168 blocks in 11 files in SY:[200,57]


Ready
```

The elements of the directory are described in the remaining sections of this chapter, "Filenames and Extensions," "Protection Codes," and "Account and File Statistics."

### 2.4.1 Filenames and Extensions

Reading the directory from left to right, one notices first each file's filename and extension, which are separated by a period (.), and appear under the headings "Name" and ".Ext". The filename and extension may be compared, with some justification, to the two parts of a person's name. The filename, or "first name," is unique in that it distinguishes the individual, or file, from others having the same extension or "family name." Thus, one knows that the first two files in the directory, AVERAG and PERCNT, are separate and distinct members of a family or class of files identified by the extension .BAS. This extension, as the reader may have supposed, in turn identifies the files' class as comprising only files written in the BASIC-PLUS language. Thus, AVERAG.BAS and PERCNT.BAS are the names of two BASIC-PLUS programs. Moreover, considering their filenames, one might infer that the user has named the first to indicate that its function is to calculate an average, and the second to indicate that its function is to calculate a percentage.

The user has thus abbreviated the identifying words because RSTS/E allows a maximum of six alphanumeric characters in a filename. The maximum allowed for an extension is three alphanumeric characters.

The user, as mentioned previously, has provided the filenames AVERAG and PERCNT. Probably, however, he did not provide their common extension .BAS. Rather, he allowed the system to supply this extension automatically, as it often does when the user creates a file on the RSTS/E BASIC-PLUS system (described fully in Part III) and does not supply an extension. Thus, the RSTS/E system, as well as the user, recognizes the significance of the default extension .BAS – i.e., that it labels a file as a BASIC-PLUS source program. There are other extensions whose meaning the system recognizes; one of these, .CMD, appears in the directory example. It tells the system that this file contains commands, and can automatically control a program – in this case, the program called BACKUP.

After the first two .BAS files, the reader encounters some files whose extensions announce that they are not members of the "family" of BASIC-PLUS programs, but belong to other identifiable classes or types of files. TERM.DOC, for example, may be a document of some kind – perhaps, as the filename suggests, a set of instructions on operating a terminal. PHONE.DIR is almost certainly a telephone directory – perhaps of people in the user's department. SOURCE.DAT has a commonly used extension, which indicates that this file contains data of some kind. The last file, VISITS.LST, may be a listing of visitors to the RSTS/E site. Thus, a filename and extension, which together are often called the file specification, can indicate generally the nature and function of the file to which they are applied.

Because a user may eventually accumulate quite a number of files, of several types, in his directory, it is suggested that he devise a system of descriptive nomenclature, and use it consistently in order to prevent confusion. The names that appear in this sample directory are typical of those used at many RSTS/E sites.

The reader should note the following rules for creating RSTS/E file specifications:

1. Only alphabetic and numeric characters (alphanumerics) are allowed in the filename and extension. Embedded spaces or tabs are not allowed.
2. A filename must contain from one to six characters.
3. Any extension specified must begin with a dot, and must contain from one to three characters.
4. A null or blank extension is permitted, in which case the dot is included in the file specification, but the extension itself is omitted.

## 2.4.2 Protection Codes

The reader will notice, in each entry of the directory and under the heading "Prot," a number in angle brackets < >. This number is called the protection code, or sometimes just the protection for brevity. Every file is assigned its own protection code, either by the system (as a default) or by the user. The function of the protection code is to prevent other users from renaming, deleting, changing, or even reading the file. As is evident in the following detail from the sample directory, files may have different protection codes. The differences determine degrees or levels of protection.

```
CHOICE.BAS     2   < 60>   28-Oct-76
SOURCE.DAT    20   < 16>   28-Oct-76
BACKUP.CMD     1   < 60>   28-Oct-76
XX    .TST     1   < 60>   28-Oct-76
VISITS.LST    14   < 40>   28-Oct-76
                     ↑─────────────────(a)
```

Verbally, one can express a file's level of protection in terms of two variables: the types of users from whom the file is protected, and the actions – reading and writing – from which it is protected. For purposes of file protection, the system recognizes three classes of users, identified by their project-programmer numbers:

1. The owner (the creator of the file)
2. The owner's group, composed of all users having the same project number as the owner
3. All other users not in the owner's group

Since these two variables – read/write privileges and class of user – determine protection, files are protected by a number of combinations. One file, for example, might be "write protected" against the owner's project group; that is, other users with his project number could read the file, but could not rename it, write anything into it, change any information presently there, or delete the entire file. Another file might be "read protected" as well as "write protected" against this same group, whose members would then be unable even to view the file's contents at their terminals. Still another file might be "write protected" only against users who do not share the owner's project number; this file could be read by any user at the RSTS/E site, but could be modified only by the owner and members of his project group.

These two combined criteria for file protection – read/write access and user class – are designated and recognized numerically by the system, in the form of the protection code. Table 2-1 lists and describes the numerical protections and their equivalents:

**Table 2-1 Protection Codes**

| If the code is: | Then the file is: |
|---|---|
| 1 | read protected against owner |
| 2 | write protected against owner |
| 4 | read protected against all others in owner's project group |
| 8 | write protected against all others in owner's project group |
| 16 | read protected against all others who do not have owner's project number |
| 32 | write protected against all others who do not have owner's project number |
| 64 | an executable program; if the file has protection <64>, the other codes (1 to 32 above) have different meanings (see Chapter 12, Table 12-4) |
| 128 | an executable program with temporary privileges |

Typically, a file's protection code is a decimal number that is the sum of the desired combination of the values in Table 2-1. In the sample directory, for instance, the most common protection code is <60>, or the sum of the following codes as listed in the table:

| | | |
|---|---|---|
| 32 | = | write protection against users without the owner's project number |
| + 16 | = | read protection against the group above |
| + 8 | = | write protection against the owner's project group |
| + 4 | = | read protection against the group above |
| = <60> | | |

Thus, the file AVERAG.BAS, whose protection code is <60>, can be neither read nor written into by anyone except its owner; that is, the user who created it. It may be of some interest to the reader that <60>, on this particular system, happens to be the default – the protection that the system automatically gives a file when the user creates one without specifying a protection of his own choice. Thus, the system may be said to respect the individual user's privacy to a high degree, automatically protecting the mysteries of his files against all others, including members of his own project.

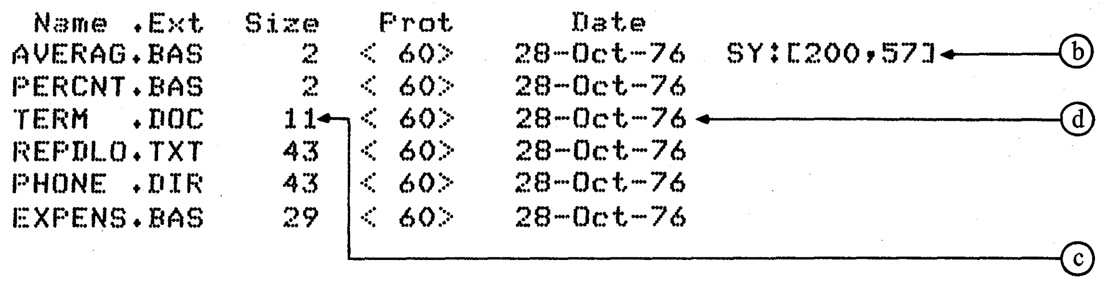
Looking at the item labelled a on the detail of the sample directory, the reader notes that the file VISITS.LST has a protection code of <40>, or the sum of the values 32 (write protection against non-members of the owner's project group) and 8 (write protection against the owner's project group). VISITS.LST, therefore, cannot be written into or modified by anyone except its owner, although any user at the site may read its information. If this file contains – as has been suggested – a listing of visitors to the RSTS/E site, then one might speculate further on its owner's reasons for thus liberating its access beyond that of the default. If the approximate number and the identities of visitors are common knowledge at the site, there would seem to be little reason to protect the listing from anyone's eyes. But if the creator of VISITS.LST has sole authority in determining who shall be considered a "visitor" (as opposed, perhaps, to a client, candidate for employment, or intruder), he would have quite a good reason to protect his file, VISITS.LST, from being capriciously edited or amended by other users. Thus, one appreciates some of the reasons for the variety of protection codes that may be assigned. The following are some typical protection codes and their meanings:

| | | |
|---|---|---|
| <60> | 32+16+8+4 | read and write protection against everyone but owner |
| <48> | 32+16 | read and write protection against all who do not have owner's project number |
| <40> | 32+8 | write protection against all but owner |
| <42> | 32+8+2 | write protection against all including owner |
| <0> | | no protection at all (any user may read and write) |

For executable programs, 64 is added to the above codes.

## 2.4.3 Account and File Statistics

The user's directory, in addition to listing the names of his files and their protection codes, also lists information about the individual and collective status of files under the user's account. This information tells both the user and the system who "owns" the files, when they were created, how many there are, and how much storage space they occupy.



```
Name  .Ext   Size    Prot       Date
AVERAG.BAS      2   < 60>    28-Oct-76   SY:[200,57]
PERCNT.BAS      2   < 60>    28-Oct-76
TERM  .DOC     11   < 60>    28-Oct-76
REPDLO.TXT     43   < 60>    28-Oct-76
PHONE .DIR     43   < 60>    28-Oct-76
EXPENS.BAS     29   < 60>    28-Oct-76
```

First, the item labeled b at the right of the directory, SY: [200,57] , indicates that all the files listed are on the public structure (identified by SY:) and "belong" to the user who holds account (or project-programmer) number [200,57] .

Within the individual file listings, the number appearing under the heading "Size" refers to the number of blocks that each file occupies on the disk. On the RSTS/E system, a block is equal to 512 bytes of 8 bits each. The file TERM.DOC for instance, as the item labeled c indicates, occupies 11 blocks of disk space. PHONE.DIR occupies 43 blocks, EXPENS.BAS 29 blocks, and so on.

A summary line at the end or bottom of the directory tells the user how many blocks are occupied by all the files together, how many files there are, and – once again – to what account number they "belong." In this sample directory, the summary line looks like this:

Total of 169 blocks in 11 files in SY: [200,57]

Finally, note the date that appears at the end of each file listing. This date indicates when the file was created. As the date d indicates, TERM.DOC was created on October 26, 1976. The same is true of the other files.

# CHAPTER 3
# THE SYSTEM RESOURCES

## 3.1 INTRODUCTION TO THE RSTS/E DEVICES

Without a set of devices at their disposal, users would be unable to print directories, obtain access to files, write programs, or even log into the system. In short, they would be unable to accomplish any work with the computer, because they would have no way to input their data, or to cause it to be output once it has been processed. The RSTS/E devices, therefore, are the vital media by which user and computer write and read information, and communicate it to one another.

The terminal is a familiar example of a device. By typing at the terminal, the user transmits information to the computer. If, for instance, this information consists of a project-programmer number and password, the computer acts on it by logging in a new job. A program can also make use of a terminal by causing it to print text and by accepting input data from its keyboard.

Thus, these media of communication between human being and computer, because of the kinds of data transfer they help to perform, are often called I/O (for Input and/or Output) devices. And, as that abbreviation suggests, some are used for input and output, some for input only, and some for output only. In the procedure just described, for example, the terminal functions as an input-and-output device: input, because the user employs it to write information into the system; and output, because the system employs it to write out an appropriate response to the user.

Another previously described procedure, asking for a directory, illustrates the I/O function of the disk as well as that of the terminal. The user types DIR at his terminal. The system, recognizing this command, reacts to it by scanning the disk for the user's directory. Once the system finds the directory, it inputs it from the disk into memory. Finally, it outputs the directory to the user's terminal.

While the terminal and disk can both be used for input and for output, some devices can be used for only one of these functions. The paper tape reader and the card reader are input-only devices. The computer can use them to "read in" a user's input from punched tape or cards, but not to "write out" its own processed output. The line printer, however, is an output only device; the computer can use it to "write out" (print) its processed output, but not to "read in" the user's input.

Because the RSTS/E system affords its users some degree of choice in I/O devices, one should be aware of the functional differences among them, and of the limitations and advantages of each. A user who knows the devices can easily decide which one best suits his current informational needs. For example, a user may wish to produce a printed copy of LOTS.TXT, a file that contains 100 blocks of text: clearly, a voluminous body of data. This user can, of course, request that the system use the terminal for output. But if he does, he is in for a long and probably a frustrating wait while the terminal labors to print all 50,000 characters a character at a time. This user would be far better advised to choose the line printer for output, since this device is specifically designed for fast printing. Moreover, while the line printer is printing his file, the user is able to employ his terminal for input of other information.

Experienced users of the RSTS/E timesharing system often make such I/O choices, preferring one device over another because of its greater efficiency or convenience. At a typical site, for example, one might find a user who has written (input) the BASIC-PLUS program MYFIL.BAS onto the public disk structure, the set of disks that is timeshared by all the other current users. Now assume that this user wishes literally to take the program away with him, in order to run it on another RSTS/E system three hundred miles away. Obviously, he may not remove a disk from the public structure and carry it off; it contains the programs and data of many other users. Even if he could somehow procure a private disk pack, one intended for his use alone, he would find it a heavy and awkward traveling companion: it can hardly be carried in a backpack or mounted on a bicycle, to say nothing of the suspicion it would arouse among airline personnel and passengers. What, then, is this itinerant user to do? One answer is to have the system copy (output) MYFIL.BAS onto a DECtape, a small I/O device consisting of a 260-foot magnetic tape wound on a plastic spool. Clearly, this device, measuring about .75 by 3.75 inches in diameter, would solve the problem of portability. Once this user arrives at the distant RSTS/E site, he may have the computer first copy (output) MYFIL onto its own public structure, from which the user may then run the program.

Meanwhile, back at the home site, another user may need to air-mail her FORTRAN program IFILE.FOR to an overseas RSTS/E site; to save postal costs and prevent damage, she copies (outputs) the program from the public structure – where it currently resides – to another device, the paper tape punch. This device "writes" the program, in the form of coded perforations, on a paper tape, a long, accordion-folded strip of paper that, obviously, is lighter in weight and less fragile than a spooled tape. Once the paper tape arrives at its destination, programmers there may first input IFILE by using the paper tape reader (an input-only device), then have their system output the program to the public disk structure.

Thus, a number of specialized Input/Output operations – reading, writing, copying – are carried on by user and computer with the help of devices, each with its own unique capabilities and limitations.

Figure 3-1 illustrates these devices as they might be configured – put together and linked to the computer to form a system – at a typical RSTS/E site. Note that the machine labeled "computer" is also called the "processor," since it computes or processes the data. When many users speak of the "computer," they actually mean the processor. Sometimes, it is called the CPU (Central Processing Unit).

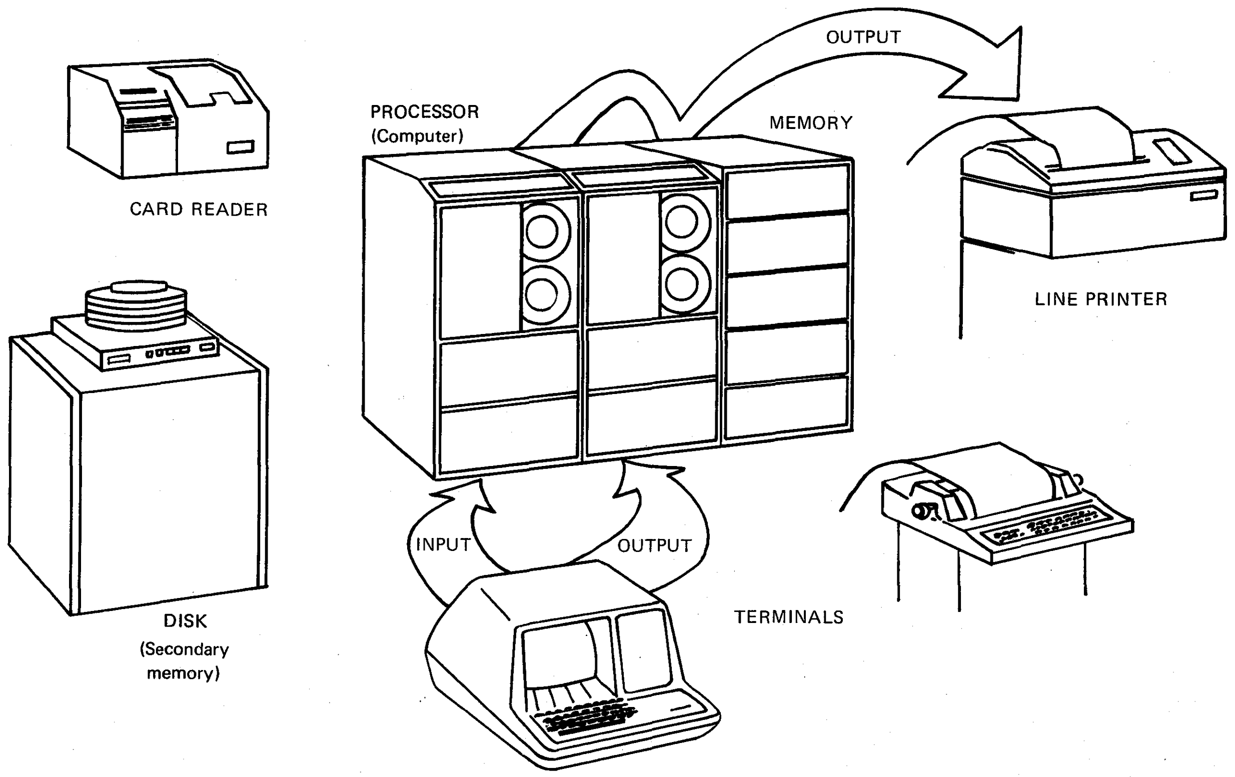


Figure 3-1 A Configuration of Devices

## 3.2 THE PUBLIC DISK STRUCTURE

As discussed in Chapter 1, "Timesharing and RSTS/E," users share computing time on the RSTS/E system, each being allotted a "slice" of the processor's time. The users may share another system resource as well: the array of devices. In the foregoing section, it was noted that one set of devices (disks) is shared by a number of users simultaneously. This shared set of devices is called the public disk structure, because it is always accessible to all users and because the system treats it as a unit. On some systems, it may include many separate disk packs. One of these, the system disk, contains the system code, language processors, and, possibly, the library of system programs, some of which are described in Part IV of this manual. The other disk packs (the public disks) contain the directories and files of the users. (The system disk too may contain some user directories and files in addition to its system information.)

The user who is logged into the system and working with a disk file is usually unconcerned about which disk in the public structure happens to contain that file. The particular disk has been chosen by the system according to current timesharing needs. Each of the disks contains a master list of users' accounts, and, for each account, a list of all files stored under that account. The system, therefore, by using these lists, is able to locate a user's file when he requests it from his terminal.

This location process can be simply explained by returning to an example of system operation discussed in the preceding section: requesting a directory. A logged-in user types DIR, and thereby asks the system to print his file directory at his terminal. Once the system recognizes this command, it responds by running the DIRECT program, which scans the master list of accounts on each disk. The master list discloses the location of the user's files – his directory. Having found the directory, the DIRECT program prints it at the user's terminal.

Another frequent user action – listing a stored program at the terminal – further illustrates the operation of the public structure. In this case, the user is logged in under account [100,105] and has been working on one file for some time – perhaps modifying a BASIC-PLUS program. This user now completes her modifications, checks the revised file, and saves it. Before logging out, however, she wishes to edit another one of her BASIC-PLUS programs: MYFILE.BAS, a file containing 30 or so lines: brief enough to be quickly and conveniently printed at the terminal. Before she can edit the program, she must take several steps: she must tell the system that she wishes to access an "old" (previously stored) program, she must tell it the program's filename, and finally she must tell it to list the program at her terminal.

She begins by typing the command OLD; when the system prompts her with OLD FILE NAME--, she types MYFILE (the system assumes the extension .BAS). Once the system receives the filename, it locates the file, scanning each public disk's master list of accounts in an effort to locate the user's file directory on that disk. Thus, the system is guided through the public disk structure to the user file directory [100,105], and ultimately to the location of MYFILE.BAS. Now the user, seeing the READY prompt at her terminal, actually causes the file to be listed at the terminal. She does so by typing LIST, whereupon the system, understanding this as a request that it list the current program, MYFILE.BAS, prints (outputs) the file at the user's terminal. Now the user may read the file and modify it as she wishes.

## 3.3 PRIVATE DISKS

A good deal of system file activity – such as creation, access, editing, and deletion – takes place on the public disk structure. Since it is the largest constantly available medium of file storage, it is generally the busiest. But not all the disks used on a system need be in the public structure. Some of them may be private disks, disk packs or cartridges that "belong" to a single user account or perhaps to a few user accounts, in the sense that these accounts alone are on the disk(s). Only if a private disk already contains an account can files be created under that account on the disk. Users without one of these private disk accounts can read or edit a private disk file, but only if its protection code permits.

For example, assume that a private disk mounted on drive DK3: belongs only to the members of a specific project group, to those users with project number 200. In such a case, the users who hold account numbers [200,30], [200,31], [200,32], [200,33], etc. may all create files on private disk DK3:. A user with account [210,33], however, may not create files on DK3:, although, protection codes permitting, he may read or edit files already created on that disk.

From the user's point of view, then, one important difference between a private disk and one on the public structure is that he may always create a file on the public disk, whereas he may do so on a private disk only if he has an account number there. On both types of disk, file protection codes govern his read and write access to existing files. Another difference, which might concern him less, is that a private disk can be mounted or dismounted while the system is running, whereas a public disk must always be mounted during timesharing. This difference, obviously, is of special concern to the owner(s) of the private disk and to the system manager, who, it should be noted, determines which disks on the system are public and which are private.

## 3.4 ASSIGNABLE DEVICES

Disks, public and private, are generally not assignable: that is, a user may not, except in rare and special cases, request one for his exclusive, temporary use; even a private disk is usually shared by a number of users. To satisfy the individual user's frequent need for input and output media devoted to his work alone, RSTS/E provides an assortment of assignable devices. DECtapes and magtapes, for example, are assignable by the individual user for his own input or output; card punches and paper tape punches are assignable for his output. Their assignability depends, of course, on their physical availability: obviously, if a DECtape or a card punch is being used by one person, another cannot immediately assign it to himself; should he try, the system will print an appropriate message at his terminal (DEVICE NOT AVAILABLE, for example).

Within this class of assignable devices, there are degrees of accessibility. A DECtape, for example, with a single directory and no accounts, permits a user access to all the files it contains. A magtape, on the other hand, which contains account numbers associated with its files, permits a user access to any files if he knows their account numbers. Paper tape punches and card punches, being unit record devices, contain no files and therefore impose no such restrictions on access; nor do line printers, which are also unit record devices. If nobody else is using a punch or printer, and if the system manager has not restricted its availability, a user is free to assign it. And anyone may use an unoccupied terminal – possibly even a person who lacks an account number and password and is therefore not a recognized user, since there are commands he can successfully give without first logging in.

## 3.5 A LIST OF RSTS/E DEVICES

The following list contains brief descriptions of devices available on RSTS/E systems, including their general functions, advantages, and disadvantages. Following the full name of each device are two items; the abbreviated name, or the specification, by which it is known to the system, and its I/O function (input and output, input only, or output only).

Devices are also classified as file structured or non-file structured; a file structured device can store files, while a non-file structured device cannot. However, a file structured device can be treated by the system as non-file structured. This capability allows an experienced user to bypass some of the limitations of file structured devices.

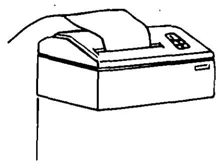
The devices are listed according to degree of legibility: those specifically intended for human reading appear first; next come those which, with some effort, are humanly decipherable; finally come those which only the computer can "read."

TERMINAL KB: or TT: or TI:
(input and output)

The terminal is a non-file structured device. In addition to human readability, its most significant advantage is that it is interactive; that is, it allows the user to communicate with the system and thus to control his job while it is running. Another obvious advantage of the terminal is its operational similarity to a common off-line, non-file structured, input-only device: the typewriter. A user who can type will have no trouble in mastering the mechanics of operating the terminal.

A disadvantage of the terminal is its inconvenience for output of large amounts of data. Compared with a line printer, a hard-copy terminal prints quite slowly. And though a video terminal may print rapidly, it may not produce a paper copy that the user may remove and read at his own pace. (Some video terminals are equipped with hard-copy devices, but these print slowly.)

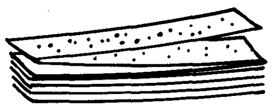
LINE PRINTER LP:
(output)



The line printer is a non-file structured device. Like the terminal, it produces human-readable output, but at much greater speed. This speed is its most important advantage over the terminal; the line printer is the fastest available device for producing hard copies of information.

The line printer's relative disadvantages arise from its singular purpose: simply to output information, and that only in the form of a human-readable hard copy (a listing). Its output therefore, unlike that of tapes, cards, and disks, cannot be "read" by the computer: there is no way to recycle one of its file listings through the system. Therefore, if the printed file has been deleted from the machine-readable device (disk, DECtape, etc.) on which it resided, the user has no way to edit it by computer. He must resort to manual editing: handwriting, "mark-up," cutting, pasting, typing, etc. And, if that deleted file was a program, he obviously cannot run it. Another disadvantage of the line printer is its inability to output more than one user's data concurrently, in the manner of the disk. Before assigning a busy line printer, a user must wait for it to finish its current job.

PAPER TAPE PR: and PP:
(input and output)



Paper tape is a non-file structured device. It has three significant advantages over other machine-readable media: it is inexpensive, easily shipped or mailed, and can be deciphered by a person who knows its punched code.

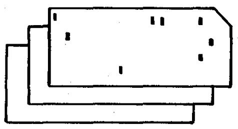
Most of paper tape's disadvantages are inherent in its physical makeup. Holes punched in paper cannot be "erased" or satisfactorily repaired: thus, a paper tape must always contain the same data; changing or editing requires a new tape. Also, paper is a low-density storage medium: a good deal of it is required to hold information in any form – even as tiny perforations. And paper, of course, is easily torn or creased.

In addition to its intrinsic disadvantages, paper tape, for full usability, requires two additional devices, one for input and one for output: namely, the reader and the punch. These are described below:

PAPER TAPE READER PR: (input)
The paper tape reader is a non-file structured device. Its advantages are that it is compact and performs automatic input that is faster than user input from a terminal. Its input speed, however, is much slower than that of DECtape, magtape, and disk. And it requires a good deal more intervention by the user.

PAPER TAPE PUNCH PP: (output)
The paper tape punch is a non-file structured device. Its important advantages and disadvantages for output are the same as those of the paper tape reader for input.

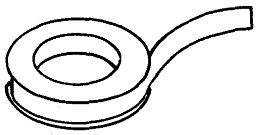
CARDS CR: and CD:
(input and output)



Cards are non-file structured. Like paper tape, they are machine readable but humanly decipherable. Because they are inexpensive, easy to mail individually or in small quantities, and can be coded or annotated with a pencil, they are widely used to gather data for public and commercial operations: school registrations, consumer billings and surveys, and so on. Cards can be prepared off-line (by keypunch), thus conserving system time and resources.

"Do not bend, fold, spindle, or mutilate." This admonition, part of contemporary folklore, succinctly expresses a major disadvantage of cards: their susceptibility to damage. Composed of card paper, they also share all the disadvantages of paper tape. For full usability, cards require two additional devices: the CARD READER (CR: and CD:) and CARD PUNCH.

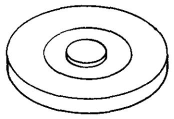
DECtape DT:
(input and output)



The DECtape is a file structured device. Its significant advantages over paper storage devices are its far greater I/O speed, its higher density of data storage, and its reusability. Unlike paper tape or cards, it can be erased, edited, and rewritten. And compared with the paper media, it involves less handling by the user, because it requires only one additional device to fulfill its I/O capability: the DECtape DRIVE, which performs both input and output. The DECtape, because it has a directory structure, also allows a user to change files in place, and therefore requires less manipulation by the system than does a magtape. Because of its small size, the DECtape is easily handled, carried, and stored. It is physically stronger and less sensitive to climate than cards, paper tape, and magtape.

The DECtape's small size presents one disadvantage: it cannot store as much data as a magtape or disk. Also, its storage density is less than that of a disk. And since the DECtape is a magnetic storage device, it cannot be humanly decoded.

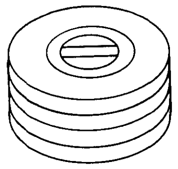
MAGTAPE MT: or MM:
(input and output)



The magtape is a file structured device. Unlike DECtape, it has no directory; it does, however, contain an individual account associated with each file on the tape. It shares with DECtape the following advantages over paper storage devices: greater speed, higher density, reusability, and less handling by the user. Also, it requires only one other device for both input and output: the MAGTAPE DRIVE. The magtape's size and shape make it convenient for storing system files off-line.

The magtape shares two relative disadvantages with the DECtape: its speed and density are not as great as those of the disk. And, as noted, it is somewhat more susceptible to damage than DECtape, and requires more system manipulation. Also, the magtape is a sequential medium: in order to reach a specific file on the tape, the system must first read all the files preceding it. And a specific magtape file cannot be deleted without also deleting all the files that follow it.

DISK SY:, DF:, DK:, DP:, DB:, DM:, or DS:
(input and output)



The disk is a file structured device. By now, the reader is probably aware of its advantages: of all devices, it is fastest, highest in density, largest, most reliable, and most durable. Like DECtape and magtape, it is reusable. And it far surpasses those devices in the number of accounts and files it can hold. For input and output, the disk requires one additional device: the DISK DRIVE. The disk involves less human handling than any other device. For all these reasons, it is chosen for the RSTS/E public structure.

The disk's obvious disadvantages are its large size, heaviness, and high cost. These make it less practical for off-line storage, for travel, and for ownership by a single user. And since the disk – like DECtape and magtape – is a magnetic device, it is not humanly decodable.

## 3.6 DEVICE NAMES: PHYSICAL AND LOGICAL

### 3.6.1 Physical Device Names

In the foregoing list of RSTS/E devices, each device's specification, the name by which it is known to the system, appears beside the device's full name. For DECtape, for example, the system's name is DT:. Such a specification, also called a physical name, is generally followed by a decimal unit number, which, in the case of a storage medium, is the number of the drive on which the medium is mounted. This number, therefore, serves to distinguish devices of the same kind according to their physical locations on the system.

For example, three users may be simultaneously creating files on three DECtapes, physically named DT0:, DT1:, and DT2:. These physical device names separate the three DECtapes – from the point of view of each user and of the system.

To elaborate on this example, one of these three users, the current "owner" of DT0:, is creating on that device a text file named NANCY.TXT. From the system's point of view, that file's more specific name, or file specification, is DT0:NANCY.TXT – the filename and extension, preceded by the physical name of the device on which the file resides. The standard colon (:), about which the reader may have been wondering, thus serves not only to identify the device name, but also to separate and to distinguish it from the filename.

A device name, in a file specification, performs the important function of identifying a file unequivocally. For example, assume that a DECtape "owner" is working on a file whose specification is DT1:FIRST.BAS and that this user has ASSIGNed the device DT1:. This user, furthermore, already has on his account an "old" or existing file of the same name – not on DECtape but on the public disk structure, a resource he has not been using because he has been working on the DECtape instead. But now he closes his DECtape file and returns to the public structure, without first relinquishing his "ownership" of the DECtape DT1: via the DEASSIGN command. Thus, although he is sharing the public disk structure with a number of current users, he still "owns" the DECtape DT1:, because he has not revoked his original device assignment command, ASSIGN DT1:. (The ASSIGN and DEASSIGN commands are explained in Part II.)

Sometime later, this user wishes to reopen and edit the file DT1:FIRST.BAS – that is, the FIRST.BAS that resides on "his" DECtape, not the FIRST.BAS that resides on the public structure. If he were to forget the physical device name DT1: in specifying the file, and simply type FIRST.BAS, he would summon not the DECtape but the disk file, and with it ample possibility for confusion. The system will retrieve the disk file FIRST.BAS because the system always assumes the public disk structure as the default device.

A RSTS/E user should, of course, protect himself against this sort of confusion by being careful when specifying duplicate filenames. The system, fortunately, cannot be so confused: it always "remembers" files not only by their names and extensions, but by their host devices as well. Thus, an individual user may have created, at various times, a number of files on devices other than public disks. Depending on the drives running these devices, the RSTS/E system would "know" and recognize these files by the following specifications: DT2:DATA3.REP, DT3:INDEX.001, MT1:ARTHUR.NAA (a magtape file), and DK5:PRIVAT.GRP (a private disk file).

### 3.6.2 Logical Device Names

Any RSTS/E device can have, in addition to its physical name, a user-assigned logical name. In other words, a user, if he so desires, may give to a device a name of his own choosing. This name, like a filename, can contain from one to six alphanumeric characters, without embedded nulls, tabs or spaces, and including the standard colon separator (:). By issuing a variation of the ASSIGN command, a user may, for example, assign to the logical name INDEX: to DECtape DT3:. This logical name will be recognized by the system until the logical name is DEASSIGNed by the user.

The reader may wonder at this point why a logical name would be used. There are several good reasons for logical naming. An obvious one is implied in the hypothetical name INDEX:; it seems to describe the nature of the information residing on DT3:, and therefore may be easier for the user to remember than the physical name with its unit number 3.

But there are more specific reasons for logical names. The fundamental advantage of a logical name is that, unlike a physical name, it does not depend upon where – i.e., on what drive – the medium is mounted. Therefore, a user may choose a logical name without regard to what device drives may be available at some future time. For example, a user writes a BASIC-PLUS program that, at several points, accesses a specific magtape for statistics. In her program, the user refers to this magtape by the logical name STATS. The advantage of STATS over a physical name is this: if the user were to specify a physical name such as MT1: for the magtape, the success of her program, as written, would depend on the availability of magtape drive #1 at the time she wishes to run the program. What if the drive were in use or inoperative – "down" – at the time? If another drive, say #2, were available, she could use it, but would first have to edit her program accordingly, changing each occurrence of "MT1:" to "MT2:" – a tedious procedure. So by assigning "her" magtape the logical name STATS before running her program, she ensures the system's recognition and access of that magtape whether it is mounted on drive #1, #2, #3, etc.

A similar use of logical names is to enhance the efficiency of a batch job, an operation which does not require terminal interaction. Typically, such a job is "programmed" by one user for later execution by another. For instance, a programmer creates a large batch job in the morning to be run by an operator at night, when there are fewer users on the system. The batch "program" or control file, like the BASIC-PLUS program in the preceding example, accesses a magtape. If the programmer refers to the tape by a logical name in his batch control file, and informs the operator of this name, he need not be concerned about the availability of a specific magtape drive. Any one will do, since the system will recognize the logical name once the operator assigns it.

About device names in general, it should be noted that the system always recognizes and accepts a device's physical name, whether or not a logical name has been assigned to that device. Thus, a DECtape running on drive #2 and assigned the logical name STAR can be specified by the user as DT2: or as STAR. Also, some logical names, at the system manager's discretion, can be made system-wide: known to, and usable by, all other users. One such system-wide logical name is SY:, which designates the public disk structure. Since SY: is a logical name for a set of devices, it need not have a unit number, and would probably be confusing if it did. SY0:, however, is always the system disk.

# PART II

# ADAPTING SYSTEM RESOURCES: DEVICES AND DEFAULTS

# CHAPTER 4
# FUNCTIONS OF THE RESOURCE COMMANDS

This part of the User's Guide contains a set of commands which apply and adapt system resources such as devices and accounts to the needs of the individual user. Among the functions performed by these commands are device assignment and deassignment, logical naming of devices and accounts, and changing protection of files. Thus, the resource commands enable the user to make full use of available hardware and of services.[1]

The resource adaptation commands have a format that resembles English grammar. The reader will note that most of these commands are English verbs: ASSIGN, DEASSIGN, and REASSIGN. And as verbs, they often specify objects of their actions. These objects can be device specifications, protection codes, accounts, etc. Therefore, the command string

```
ASSIGN DT0:
```

which consists of the command ASSIGN followed by the device specification DT0:, tells the system to reserve DECtape unit 0 for the user who has given the command.

Sometimes, a resource command need not specify an object of its action, either because the object is understood by the system, or because an object is not needed to complete the command's meaning. DEASSIGN used alone, for instance, tells the system to release all devices from the user's control. And TAPE, which never needs an object, tells the system to disable the terminal echo feature.

Table 4-1 is an overview of the resource commands – a guide to their functions and their locations within Part II:

**Table 4-1 A Guide to the Resource Commands**

| | Command & Format | Section |
|---|---|---|
| PHYSICAL DEVICES | | |
| Reserving | ASSIGN dev: | 5.1 |
| Releasing | DEASSIGN<br>DEASSIGN dev: | 5.2 |
| Transferring control | REASSIGN dev:job-number | 5.3 |
| Disabling terminal echo | TAPE | 5.8.1 |
| Enabling terminal echo | KEY | 5.8.2 |
| LOGICAL NAMES | | |
| Associating with device | ASSIGN dev:logical-name | 5.4<br>5.4.1<br>5.4.2 |

[1] Before attempting to issue commands, the user should check to see if his terminal has been set in BASIC-PLUS EXTEND mode, in which spaces and tabs are significant (see Section 9.2.1).

**Table 4-1 (Cont.) A Guide to the Resource Commands**

| | Command & Format | Section |
|---|---|---|
| LOGICAL NAMES (Cont.) | | |
| Associating with account | ASSIGN [proj,prog] | 5.7 |
| Cancelling logical association | DEASSIGN logical-name | 5.4.4 |
| | DEASSIGN@<br>DEASSIGN[proj,prog] | 5.7 |
| System-wide logical names | | 5.5 |
| Pack identification label | | 5.6.1 |
| LOGICALLY NAMED DEVICES | | |
| Reserving | ASSIGN logical-name: | 5.4.3 |
| Releasing | DEASSIGN logical-name: | 5.4.3 |
| Logically named disk pack or DECpack cartridge | | 5.6.2 |
| Logically mounting disk pack or DECpack cartridge | | 5.6.1.1 |
| CHANGING DEFAULTS | | |
| Changing protection default | ASSIGN <prot> | 6.1 |
| Changing magtape labeling default | ASSIGN MTn:.label | 6.2 |

# CHAPTER 5
# CONTROLLING DEVICES AND ACCOUNTS

## 5.1 RESERVING A DEVICE: THE ASSIGN COMMAND

The ASSIGN command reserves an I/O device for the use of one programmer (i.e., one job number).

To reserve a device for his exclusive use, the user types the command ASSIGN and an object, in this form:

```
ASSIGN dev:
```

The object dev: is a device specification. (For a list of possible specifications, see Table 5-1.) If the device is available for use, the system responds by printing the prompt

```
READY
```

The user, seeing the READY prompt, may then perform I/O with the assigned device.

If the device is not available for use, the system responds with the message

```
?DEVICE NOT AVAILABLE
```

There are several reasons for a device's unavailability; it may be 1) opened or assigned by another user, 2) reserved for a specific operation (a terminal may be in use as a pseudo keyboard), 3) restricted by the system manager for privileged use or hardware maintenance.

If the device is not even configured on the system, the user receives the error message ?NOT A VALID DEVICE.

In the following example, a user successfully assigns line printer 0, but tries unsuccessfully to assign the high-speed paper tape reader, because that device is unavailable. (The user's input is underlined.)

```
ASSIGN LP:
READY

ASSIGN PR:
DEVICE NOT AVAILABLE
```

If more than one job is logged into the system under a single account number, only the job (i.e., user) performing an ASSIGN (or DEASSIGN) is affected by that command. Devices reserved by a job remain in that job's control until the user who created the job releases the devices or logs off the system. Because devices are controlled by a job, the device assignments remain in effect even if the user changes accounts.

## 5.2 RELEASING A DEVICE: THE DEASSIGN COMMAND

The DEASSIGN command releases an assigned device from the user's control back to the system's supply of available devices. Thus, DEASSIGN makes the device available for use by other jobs.

Issued with no device specification, DEASSIGN releases all the user's (job number's) assigned devices. For example

```
DEASSIGN
```

releases all devices that the user has assigned under the current job number. If a user does not issue this command before logging out, the system itself performs a DEASSIGN when the user does log out.

**Table 5-1 Device Specifications**

| Specification | Device |
|---|---|
| DF:,DK:,DP:,DB:,DM:,DS:, or SY: | RSTS/E public disk structure as a whole |
| SY0: | System disk (the unit which was bootstrapped) |
| DF0: | RF11 disk |
| DK0: to DK7: | RK11/RK05 disk cartridge units 0 through 7 |
| DP0: to DP7: | RP11/RP02/RP03 disk pack units 0 through 7 |
| DB0: to DB7: | RP04 disk pack units 0 through 7 |
| DM0: to DM7: | RK611/RK06 disk cartridge |
| DS0: to DS7: | RH11/RS03/RS04 fixed head disk units 0 through 7 |
| PR: | High-speed paper tape reader |
| PP: | High-speed paper tape punch |
| CR: | CR11 punched or CM11 mark sense card reader |
| CD: | CD11 punched card reader |
| MT0: to MT7: | TM11/TU10 or TS03 magtape units 0 through 7 |
| MM0: to MM7: | TM02/TU16 or TU45 magtape units 0 through 7 |
| LP0: to LP7: | Line printer units 0 through 7 |
| DT0: to DT7: | TC11/TU56 DECtape units 0 through 7 |
| KB: | Current user terminal |
| KBn: | Terminal n in the system |
| TTn: | Terminal n in the system (synonym for KBn:) |
| TI: | Current terminal (synonym for KB:, the terminal that initiated the job) |
| DX0: to DX7: | Floppy disk units 0 to 7 |

**NOTE**

The user can reference LPn:, DTn:, DXn:, KBn:, MMn: and MTn: where n is between 0 and the maximum number of such units on the system. LP:, DT:, DX:, MM: and MT: are each the same as specifying unit 0 of the related device.

To release a specific device, the user may give the DEASSIGN command followed by a device specification. For example

```
DEASSIGN LP:
```

releases line printer unit 0.

The DEASSIGN command, when used to release a magtape, automatically causes the magtape to return to the system's labelling default.

## 5.3 TRANSFERRING A DEVICE: THE REASSIGN COMMAND

The REASSIGN command transfers control of a device to another job. For example, if DECtape unit 1 is under control of the current job, the command

```
REASSIGN DT1:8
```

transfers control of the DECtape to job number 8.

In performing this transfer between two jobs, the REASSIGN command also prevents a third job from gaining control of the device. In the foregoing example, job number 8 might be busy with an operation that prevents it from using DT1: immediately after reassignment. If job number 20, for instance, attempts to assign this DECtape before job number 8 is ready to use it, the attempt will be unsuccessful.

An attempt to reassign control of a device to a nonexistent job causes the system to print the %ILLEGAL NUMBER error. If the device is open or has an open file, the system generates the error ?ACCOUNT OR DEVICE IN USE. Before transferring control, the user must close the device, or close any files open on the device.

Magtape users should note that the labelling format, density, and parity characteristics assigned to a magtape unit are preserved in the REASSIGN command's transfer.

## 5.4 ASSIGNING AND USING LOGICAL NAMES: THE ASSIGN AND DEASSIGN COMMANDS

Logical names for devices, discussed in Section 3.6.2, are assigned by the user and do not depend on the physical device specifications. Thus, a user who writes a program referencing physical devices can give these devices logical names of his own choosing in the program. Before running the program, he can issue the ASSIGN command to associate his chosen names with the devices. This action makes the program independent of the devices' physical locations on the system.

To associate a logical name with an assignable physical device, the user types the following form of the ASSIGN command:

```
ASSIGN dev:logical name
```

where dev: is the specification of the physical device. The logical name can be from one to six alphanumeric characters. A job can have a maximum of four logical name assignments at a time. If the user attempts to make a fifth logical assignment, the system prints the error message ?ACCOUNT OR DEVICE IN USE. A logical association is unique to the job and is preserved during CHAIN operations. And because the logical assignment is job-related, it is also preserved when a user changes accounts. The command does not reserve the device but merely associates the logical and physical names.

### 5.4.1 Associating Multiple Logical Names with One Device

If the user makes two logical name assignments for the same device, the system recognizes both logical names as belonging to that device. For example:

```
ASSIGN DT1:A
READY

ASSIGN DT1:B
READY
```

As a result of these commands, the system associates both logical names A: and B: with DECtape unit 1.

If the user associates two different devices with the same logical name, the system replaces the former logical assignment with the latter assignment. For example:

```
ASSIGN DT1:A
READY

ASSIGN DT2:A
READY
```

After execution of these commands, the system associates logical name A: with DECtape unit 2.

## 5.4.2 Associating a Valid Physical Name with a Device

If the user associates a device with a valid physical device name, the system recognizes the logical, and not the physical, assignment. For example:

```
ASSIGN DT0:DT4
READY
```

The system subsequently associates the physical device designator DT4: with DECtape unit 0. The system makes this association only for the job that has assigned this logical name. When another job requests DT4:, the system will in fact attempt to access the physical device DT4:.

## 5.4.3 Reserving and Releasing a Logically Named Device

To reserve a device for which a logical name exists, the user may type the ASSIGN command, followed by the logical name and a colon. The command takes the following form:

```
ASSIGN logical name:
```

The system then reserves the associated physical device if it is available. Note that the colon is required.

The following commands reserve DECtape unit 1 and associate a logical name with that device:

```
ASSIGN DT1:
READY

ASSIGN DT1:ABC
```

As a result of these commands, a BASIC-PLUS statement of the form OPEN "ABC:FILE.EXT" AS FILE 1 in a subsequently executed BASIC-PLUS program attempts to open FILE.EXT on DECtape unit 1. Also, the subsequent use of ABC: in any system command refers to DECtape unit 1. An attempt to refer to a device by an unassigned logical name generates the error ?NOT A VALID DEVICE.

To release control of the physical device if a logical name is still in effect, the user may type the following form of the DEASSIGN command:

```
DEASSIGN logical name:
```

Note that the colon is required. This command releases control of the physical device associated with the logical name. And any device, of course, can be released by issuing DEASSIGN with a physical name. Neither of these forms of the DEASSIGN command, however, cancels the association between physical device and logical name.

## 5.5 SYSTEM-WIDE LOGICAL NAMES

At the system manager's discretion, a system-wide logical name may be associated with a device or a device and an account on the device. This name, once assigned, may be used by all jobs on the system. For example, a manager associates the logical name CUP: (for Commonly Used Programs) with account [3,4] on the disk cartridge DK3:. Subsequently, a user may run a program FOO that is under account [3,4] on DK3: by typing the command

```
RUN CUP:FOO
```

As a result of this command, the system searches for the file FOO.BAC under account [3,4] on RK05 disk unit 3.

On the other hand, the manager may associate a system-wide logical name with a physical device name only, without specifying an account on that device. In this case, the default account is that of the job accessing the device. If, for example, the disk pack DP1: has been assigned the system-wide logical name SCRACH:, a user who types the command

```
RUN SCRACH:MYFILE
```

will cause the system to look only under his account on DP1: for the program MYFILE.

The default account associated with a system-wide logical name may be overridden by specifying another account after the logical name. For example, a user whose account is [200,210] wishes to run the program OTHER from account [200,240] on disk pack SCRACH:. To do so, this user types

```
RUN SCRACH:[200,240]OTHER
```

As a result, the system searches the disk pack for the file OTHER.BAC under account [200,240] rather than under any account associated with the logical name or under the user's own account [200,210]. Thus, the account appearing after the logical name overrides the default account.

**NOTE**

If the system manager has associated an account with the system-wide logical name, the order in which device name and account are specified is significant. If the user in the preceding example were to type

```
RUN [200,240]SCRACH:OTHER
```

The system would search for the file OTHER under the account associated with the logical name. If no account were associated with the logical name, the system would in fact search for the file under the user's own account [200,240].

## 5.6 DISK ACCESS BY PACK IDENTIFICATION LABEL OR LOGICAL NAME

This section illustrates how a disk may be accessed by one of three types of names: a physical name, a system-wide logical name (e.g., a pack identification label), or a user-assigned logical name.

### 5.6.1 Disk Access by Pack Identification Label

Each disk pack or DECpack cartridge on the system has written on it a pack identification label: for example, MYPACK. This label is not a physical name, because it is independent of which drive unit currently holds the pack. But, as this section explains, the user may turn a pack identification label into a temporary system-wide logical name.

In order to access files on the disk, the user must first logically mount the disk pack – that is, establish, on the system, the association between the pack and its identification label, MYPACK. Since this association is established within the executive's tables, the label becomes a system-wide logical name for the disk pack. Logical mounting is performed when the user specifies the pack identification label in the CCL command MOUNT (see Section 20.2.1).

After logically mounting a disk pack or DECpack cartridge, the user may refer to it by its pack identification label. For example, to print a directory of the current account on MYPACK, which is logically mounted on drive unit 1, the user types the command

```
CAT MYPACK:
```

The label, because it is a system-wide logical name, allows all current jobs to refer to a disk pack or DECpack without concern about its current drive unit number. The name MYPACK is of course temporary, because the user can later logically dismount the disk pack or DECpack.

**5.6.1.1 Logically Mounting a Disk by System Command: MOUNT** – The user can logically mount a disk pack during timesharing by the MOUNT command. This command's function is similar to that of the UMOUNT library program. MOUNT, however, is not a CCL command; thus, the UMOUNT program need not be present in the system library in order for the MOUNT command to work. But if the CCL command MOUNT is installed on the system, it takes precedence over the system command MOUNT. (See Section 13.1.) There is no system command to logically dismount the disk.

To logically mount a disk pack, the user types the system command MOUNT in the following format:

```
MOUNT dev:pack id[/RO[NLY]]
```

In this format, dev: represents the device designator of the specified disk drive (DK1:, for example). The term pack id represents the pack identification label of the disk pack (MYPACK, for example). If desired, the file specification option /RONLY (Read ONLY) may be included (see Section 12.5.3).

### 5.6.2 Disk Access by Logical Name: The ASSIGN Command

The ASSIGN command, followed by 1) the pack identification label of a logically mounted disk pack or DECpack, and 2) any alphanumeric string from one to six characters long, logically associates the alphanumeric string with the pack. In other words, the string becomes a logical name for the pack. In the following example, the user first logically mounts a disk pack, then uses ASSIGN to make the logical association:

```
MOUNT DP1:MYPACK
READY

ASSIGN MYPACK:ZOOMAR
READY
```

Afterward, the user may type the logical name ZOOMAR: anytime he wishes to refer to the pack logically mounted on RP disk drive unit 1. Note that ZOOMAR: is a job-local logical name and applies only to one current job, whereas the label MYPACK is a system-wide logical name, and applies to all current jobs.

After he assigns this job-local logical name, ~~the~~ user can refer to a file FILE.EXT on the pack MYPACK (logically mounted on RP drive unit 1) in any of the following four ways:

| | |
|---|---|
| by physical device name, | DP1:FILE.EXT |
| by pack identification label, | MYPACK:FILE.EXT |
| by logical device name, | ZOOMAR:FILE.EXT |
| or by system-wide logical name. | SCRACH:FILE.EXT |

The association between the pack identification label and the physical device remains in effect until the pack is logically dismounted by the DISMOUNT command (see Section 20.2.4). The association between the logical name and the physical device remains in effect until the DEASSIGN logical name command is issued, or until the job is logged off the system.

In searching for a specified device, the system follows this procedure: first, it determines if the device name is a job-local (user-assigned) logical name; if it is not, the system determines if it is a system-wide logical name; finally, if the device name is neither of these, the system assumes that it is a physical device designator.

## 5.7 LOGICAL ASSIGNMENT OF A USER ACCOUNT

The ASSIGN command, followed by a user account number, establishes a logical association between that account and the commercial at sign (@) character. For example, the command

```
ASSIGN [100,101]
```

associates the @ character with the account [100,101]. The @ character, therefore, when used in subsequent commands and program statements, refers to account [100,101]. For example, the command CAT@ prints a directory of account [100,101]. Moreover, BASIC-PLUS statements such as

```
OPEN "[100,101]FILE.EXT" AS FILE 1
```

can be shortened in the following manner:

```
OPEN "@FILE.EXT" AS FILE 1
```

If an account has not been logically assigned, an attempt to refer to it by the @ character generates the error ?ILLEGAL FILE NAME.

To cancel the association between the @ character and the account, the user may type the DEASSIGN command in one of two forms:

```
DEASSIGN [100,101]
```

or

```
DEASSIGN @
```

The logical assignment remains in effect until the user issues the DEASSIGN @ command or until he logs off the system, or until he makes another assignment. Because the logical assignment of an account is job-related, it remains in effect even when the user changes accounts.

## 5.8 TERMINAL ECHO SETTINGS

### 5.8.1 Disabling the Terminal Echo: The TAPE Command

The TAPE command, followed by a carriage return, disables the terminal echo feature while the low-speed reader (located on some terminals) is reading a paper tape into the system. This command, in other words, stops the terminal from printing what is on the paper tape, and thus avoids meaningless output by the terminal. The user first types

```
TAPE
```

then the RETURN key. The user then inserts the tape in the low-speed reader and sets the reader's control switch to START.

Before giving the TAPE command, the user must cause the system to expect the tape input. For example, the sequence

```
NEW PROG

READY

TAPE
```

causes the system to await entry of a source program file from the terminal tape reader. Note that the system does not print READY after the TAPE command, because TAPE disables the terminal echo. The terminal echo feature is disabled so that the program is not listed on the terminal as it is read. This suppression of printing allows faster input than typing

```
NEW PROG
```

and then making the system read the tape through the low-speed reader. A program listing can be obtained on a line printer or on the terminal at a later time, if necessary.

In TAPE mode, RUBOUT key characters are ignored. RETURN and LINE FEED key characters are transmitted as is, but the LINE FEED or RETURN key characters are not appended since the next character on the input tape is the proper second character.

The TAPE command does not cause suppression of error messages.

### 5.8.2 Enabling the Terminal Echo: The KEY Command

Since no characters input from the terminal keyboard or reader are echoed following the TAPE command, the KEY command is provided to again enable the terminal echo feature. The user is advised to type the LINE FEED key before issuing the KEY command in case the last line input was not terminated with a carriage return/line feed pair. The command is typed as

```
KEY
```

and entered to the system with the LINE FEED or ESCAPE key.[1] (Carriage return characters are not treated as delimiters when the terminal is in TAPE mode.) Note, however, that the KEY command is not echoed at the terminal, because echoing has been disabled. The READY prompt, on the other hand, is echoed and indicates to the user that the terminal echo is once again in operation. Following successful entry of the KEY command, characters are again echo-printed at the terminal.

## 5.9 INPUT AND OUTPUT CONTROL CHARACTERS

The control characters described in this section are to aid the user in performing input/output operations at the terminal. A character preceded by the word "control" (abbreviated as CTRL) is typed by holding down the CTRL key, typing the character (C, for example), and releasing both keys.

### 5.9.1 CTRL/C

Typing a CTRL/C causes RSTS/E to print READY and return to command mode where commands can be given or editing done. CTRL/C stops whatever RSTS/E was doing at the time (execution or output) and returns control of the system to the user.

Note that CTRL/C interrupts processing. For example, if CTRL/C is used after the REPLACE command is given and before the READY reply is received, the file is not replaced in its entirety and is not closed. Since the REPLACE is not completed, parts of the program will be lost. Similarly, if the OLD command is issued and a list of error messages is being printed, the CTRL/C key should not be used since the current program is only half compiled at that point.

[1] ESCAPE is shown as ALT MODE on some terminals.

In most cases, typing CONT causes the program to continue execution. Some keyboard output, however, may be lost.

### 5.9.2 CTRL/O

The CTRL/O combination suppresses output to the terminal until the next time CTRL/O is typed. When a program produces a large amount of output (usually tabular form), the user may not wish to wait for the printing of the complete information. CTRL/O enables the user to monitor the output while not stopping it completely. Typing CTRL/O while output is occurring does not stop the computer's output; the terminal, however, does not print it. The second time CTRL/O is typed, the output is again printed at the terminal. Printing, however, does not resume at the point of the first CTRL/O, but at the point of the second CTRL/O; thus, some output may be skipped in the printing.

Unlike CTRL/O, CTRL/C completely terminates program output. It is useful to think of CTRL/O as a switch, whose first setting creates a condition and whose second setting releases the condition.

### 5.9.3 CTRL/S and CTRL/Q

These two control characters work together on display (CRT) terminals. CTRL/S temporarily suspends output to the display terminal. It is used to examine the lines currently displayed before they are replaced on the screen by additional lines. Output can be resumed at the next character by typing the CTRL/Q combination. The CTRL/S and /Q feature is usable only if the terminal has been initially defined with the STALL characteristic (see Section 20.1, Table 20-1; TTYSET program).

### 5.9.4 CTRL/Z

The CTRL/Z combination is used to mark the end of a data file. When data is input from a file, the CTRL/Z character marks the end of recorded data. The message ?END OF FILE ON DEVICE is printed by the system when a CTRL/Z is detected, unless an ON ERROR GOTO statement is used to enable a BASIC-PLUS routine to handle the error.

### 5.9.5 RETURN Key

The RETURN key, when typed, echoes as a carriage return/line feed operation on the terminal, as long as the terminal is not in tape mode. The RETURN key is normally used to terminate a line and enter that line to the system. In tape mode (following a TAPE command; see Section 5.8.1), all carriage returns are ignored.

### 5.9.6 ESCAPE or ALT MODE Key

The ESCAPE key, like the RETURN key, is used to terminate the current line and causes the line to be entered to the system. The ESCAPE key, however, echoes on the terminal as a $ character and does not perform a carriage return/line feed. ESCAPE is used to enter the KEY command to the system (see Section 5.8.2).

On some terminals the ESCAPE key is replaced by the ALT MODE key, which performs the same functions.

# CHAPTER 6
# CHANGING DEFAULTS

## 6.1 CHANGING THE DEFAULT PROTECTION CODE: THE ASSIGN < > COMMAND

The ASSIGN command, followed by a protection code in angle brackets < >, changes the default protection code which the system assigns to files created by the current job during timesharing. Usually, this default is <60> when the user logs into the system. To change it, the user types ASSIGN followed by the new code enclosed in angle brackets.

The following command, for example, changes the default protection to <40> and assigns that value to all files subsequently created under the job's control.

```
ASSIGN <40>
```

The default protection remains in effect until the user assigns another default protection or until he logs off the system. Because the default protection is job-related, its assignment remains in effect when the user changes accounts.

## 6.2 CHANGING THE MAGTAPE LABELING DEFAULT

The magtape labeling default is set by the system manager, and is system-wide. Though it normally remains in effect during the timesharing session, it can be changed for an individual job.

To change this default, the user types the ASSIGN command followed by 1) the magtape's physical name, and 2) either .DOS or .ANSI. An example follows:

```
ASSIGN MT0:.DOS

READY
```

As a result of this command, the system reserves unit 0 for the current job and treats files on unit 0 as having DOS labels. Similarly, to change the default to ANSI labeling, the user types ASSIGN, the physical device name, and the new default:

```
ASSIGN MT0:.ANSI

READY
```

The user should note the importance of the dot (.) character in each example. If it is omitted, the system assigns the logical name DOS or ANSI to the magtape unit. The labeling default remains in effect when the device is reassigned to another job and when the user changes accounts.

The user should also note the following caution about magtape labeling defaults. If the user intends to use a tape for ANSI files, and that tape has previously been used for DOS files, he must assign the magtape unit as .ANSI before using PIP to zero the tape. Similarly, if a tape has previously been used for ANSI files, the user must assign the unit as .DOS before zeroing it. Should the user fail in either case to assign the appropriate label – .ANSI or .DOS – he will receive the error message ?BAD DIRECTORY ON DEVICE when he attempts to zero the tape.

The DEASSIGN command automatically makes the magtape unit return to the system's labeling default.

# PART III

# THE BASIC-PLUS SYSTEM COMMANDS

[illegible]

[illegible]

# CHAPTER 7
# FUNCTIONS OF THE BASIC-PLUS SYSTEM COMMANDS

This part of the User's Guide describes the BASIC-PLUS system commands, special characters, and system features such as EXTEND format and immediate mode. Most of the commands described in this part do not belong to the BASIC-PLUS language itself, but are nonetheless needed by the BASIC-PLUS user. They enable the user to write, run, edit, save, and debug – that is, test and correct – programs. They comprise, in short, the support system for the BASIC-PLUS language. It is therefore assumed that the reader of Part III knows something of the BASIC-PLUS language – enough, at least, to write some short programs. Readers who need to learn more of the language are referred to the *BASIC-PLUS Language Manual.*

## 7.1 SOME DEFINITIONS FOR THE NEW BASIC-PLUS USER

For the user who may be newly acquainted with BASIC-PLUS and with its supporting system commands, some explanations of terms used in Part III are offered here.

### 7.1.1 Source and Compiled Programs

Most of the commands and operations described in Part III are designed to affect and manipulate a source program. Briefly, a source program is any program which can be translated by BASIC-PLUS, is stored on disk in ASCII (or human readable) format, and is available to a user for listing (printing at the terminal) and for modification (editing and debugging). Source programs are stored with .BAS file extensions, and are sometimes called BAS programs.

A source program is distinguished from a compiled program, which has already been translated by BASIC-PLUS, and has been stored in its translated form. The translated BASIC-PLUS code is called intermediate code. It is not human readable and must be executed by the BASIC-PLUS run-time system. This type of program is not available to the user for listing and modification, but only for execution. For this reason, compiled programs are sometimes called run-only programs. Compiled programs are stored with .BAC file extensions, and are sometimes called BAC programs as well.

### 7.1.2 The Program Currently in Memory

Often in Part III, a BASIC-PLUS program is described as "currently in memory" or as "the current program." These phrases both refer to a program that is presently available to the user – the program that he is writing, editing, listing, running, or debugging. This "current program" may be a new one that the user has just created during the present timesharing session, or it may be an "old" program that he has retrieved from his storage area. Similarly, it may be a source program or a compiled program. The important point to remember is that it is the program with which the user is now working.

Unless otherwise noted in an individual command description, the BASIC-PLUS system commands do not alter the current program.

## 7.2 A GUIDE TO THE BASIC-PLUS SYSTEM COMMANDS

Table 7-1 is an overview of the BASIC-PLUS system commands, features, and special characters, and to their locations within Part III.

**Table 7-1 A Guide to the BASIC-PLUS System Commands**

| Effect on Program | Command/Feature | Section |
|---|---|---|
| Calling old program | | |
| Calling NONAME | OLD | 8.3 |
| Compiling, | | |
| description of | COMPILE | 8.4.3.1 |
| Creating and | NEW | 8.1.1, 8.2.2 |
| Naming | NEW filename | 8.1.1 |
| Creating NONAME | NEW | 8.1.1.1 |
| Debugging | | |
| halting execution | STOP | 10.2.1 |
| | CTRL/C | 10.3.1 |
| | PRINT LINE | 10.3.1 |
| continuing execution | CONT | 10.2.2 |
| | CCONT(privileged) | 10.2.3 |
| suppressing/continuing | | |
| output | CTRL/O | 10.3.2 |
| | CTRL/S, CTRL/Q | 10.3.3 |
| Deleting | | |
| entire contents | | |
| specific lines | | |
| segments | DELETE | 9.1.2 |
| program from disk | | |
| program from private dev: | UNSAVE | 9.1.4.1 |
| | KILL | 9.1.4.2 |
| Device for storage | | |
| specifying | SAVE dev: | 8.2.2 |
| running from private | RUN dev: | 8.4.2 |
| removing from | UNSAVE | 9.1.4 |
| Directory (catalog) | | |
| printing at terminal | CATALOG | 8.9 |
| EXTEND/NO EXTEND | | |
| descriptions | | |
| changing formats | EXTEND/NO EXTEND | 9.2.1 |
| Formatting lines | | |
| (see also EXTEND above) | | |
| multiple statements, 1 line | : or \ | 9.2.2.1 |
| 1 statement, multiple lines | LINE FEED | 9.2.2.2 |
| spacing | space/TAB | 9.2.2.3 |

(continued on next page)

**Table 7-1 (Cont.) A Guide to the BASIC-PLUS System Commands**

| Effect on Program | Command/Feature | Section |
|---|---|---|
| Immediate mode execution | Immed. mode | 10.1 |
| Length of program | | |
| finding current | | |
| finding maximum | LENGTH | 8.7.2 |
| Line printer | | |
| obtaining output | SAVE LP: | 8.2.3 |
| Listing at terminal | | |
| whole program | | |
| specific lines | | |
| segments | LIST | 9.1.1 |
| without header | LISTNH | 9.1.1 |
| (summary of LIST[NH] commands) | . . . . . . . . | 9.1.1 |
| Listing at line printer | SAVE LP: | 8.2.3 |
| Merging programs | APPEND | 9.1.5 |
| Paper tape output | SAVE PP: | 8.2.3 |
| Protection code | | |
| changing | NAME AS | 8.7.2 |
| adding to compiled | COMPILE < prot > | 8.4.3.2 |
| Renaming | | |
| current program | SAVE filename | 8.2.1 |
| disk/DECtape file | RENAME | 8.5 |
| | NAME AS | 8.7.1 |
| Replacing saved program | REPLACE | 8.6 |
| Running | | |
| current program | | |
| old program | RUN | 8.4.1 |
| omitting header | RUNNH | 8.4.1 |
| from private dev: | RUN dev: | 8.4.2 |
| Saving | SAVE | 8.2 |
| compiled version | | |
| compiled version, renamed | COMPILE | 8.4.3.2 |
| Scaled arithmetic | SCALE | 8.10 |

(continued on next page)

**Table 7-1 (Cont.) A Guide to the BASIC-PLUS System Commands**

| Effect on Program | Command/Feature | Section |
|---|---|---|
| Stopping execution (see Debugging) | | |
| Transferring control | CHAIN | 9.1.6 |
| Writing | | |
| new program | NEW | 8.1.1, 8.1.2 |

# CHAPTER 8
# CREATING AND RUNNING A BASIC-PLUS PROGRAM

## 8.1 WRITING THE PROGRAM

### 8.1.1 The NEW Command

The NEW command allows the user to name and to create a new program. To issue this command, the user types

```
NEW
```

and presses the RETURN key. The system responds by printing the following request for the new program's name:

```
NEW FILE NAME --
```

The user then types the new program's filename, without an extension. (The system does not require an extension at this point because the SAVE and COMPILE commands, described later in this chapter, automatically append one.)

Another way to issue the NEW command is to type NEW, then the new program's name, then the RETURN key. Thus, the user avoids the NEW FILE NAME-- prompt. For example, the command

```
NEW CASH01
```

is equivalent to the following sequence:

```
NEW
NEW FILE NAME-- CASH01
```

When the NEW command is issued, it has the following effects in the user's memory area:

1. It deletes any program currently in memory.
2. It causes RSTS/E to remember the new program's name.

**NOTE**

A command of the following form is meaningless:

```
NEW DT0:STAT
```

This command is meaningless because new programs can be input from only one device: the user terminal. To input programs from other devices, the OLD command must be used.

When the user names a file in the NEW command, the system does not check for a file of the same name. This checking occurs when the user gives the SAVE command.

Whenever the user creates a program (with the NEW command) or calls an existing program (with the OLD command), the system creates the file TEMPnn.TMP in his area on the public structure; the nn represents the current job number. TEMPnn.TMP contains the ASCII text of the source program, and any revisions the user makes to that text. As its name indicates, this file is temporary and is destroyed when the user logs off the system. TEMPnn.TMP is used only by the BASIC-PLUS system, as a "scratch" file; the user is normally unaware of this file, except insofar as it occupies space in his area, and reflects the size of the current program.

**8.1.1.1 Creating a NONAME File** – Instead of specifying a filename in response to the prompt NEW FILE NAME --, the user may merely press the RETURN key. This action creates a file called NONAME. Later NONAME can be saved or compiled, and referenced as NONAME.BAS or NONAME.BAC. The user may change this name at any time (see Section 8.5 and 8.7.1). The following example shows the creation of the file NONAME (the RETURN key, although typed, does not echo):

```
NEW
NEW FILE NAME --

READY
```

If the user issues a SAVE at this point, the file NONAME.BAS will be created.

**NOTE**

NONAME, although a legal filename, should not be used for a program that the user wishes to save. Any user logged into the same account may create a NONAME file. Only the latest version of the file will be saved. Therefore, NONAME should be used only for programs which the user intends to run once.

### 8.1.2 Input of the New Program

Once the user has created a new file in memory with the NEW command, he may type the program into the system. Or, if his terminal has a low-speed reader, he may use the reader to input a program from a pre-punched paper tape. Information on using the low-speed terminal reader is found in Section 23.1.4.

If the user is typing his program into the system, he will likely use the RSTS/E editing features, described in the following chapter (Chapter 9). At this point, however, it is useful to describe two simple editing tools: RUBOUT and CTRL/U.

Should the user make a typographical error, he may erase the incorrect characters by typing the RUBOUT key (on some terminals, the DELETE key) once for each character to be erased. As each character is erased in this manner on a hard-copy terminal, it is echoed between backslashes \ \. After erasure, the user may type the correct characters on the same line.

On a video (CRT) terminal, typing the RUBOUT/DELETE key moves the cursor back one space, thus deleting the erroneous character. Neither the character nor any backslashes appear.

The user may at times wish to delete the current line entirely; it may, for instance, contain a large number of errors. To delete the current line, the user types the CTRL/U combination: that is, he holds down the CONTROL key while typing U. As a result of this action, the system deletes the entire current line, and performs a carriage return/line feed. The user may then retype the line.

## 8.2 SAVING THE PROGRAM: THE SAVE COMMAND

The primary function of the SAVE command is to store a current BASIC-PLUS source program on the public disk structure. To be SAVEd, the source program must be currently in memory. And after the program has been SAVEd, it remains in memory and therefore can be run, changed, or deleted.

To store the current program under his own account on the public structure, the user types

```
SAVE
```

then presses the RETURN key. As a result, the program currently in memory is saved on the public structure under the current account, with its current filename and the extension .BAS. If a file of the same name exists, the system prints the error message

```
FILE EXISTS – RENAME/REPLACE
```

This message, in telling the user that an identically named file exists in his area, warns him against unintentionally destroying that file. Should he in fact wish to destroy it by replacing it with the current program, he may – as the message suggests – issue a REPLACE command (described in Section 8.6). But if the user has no desire to destroy the identically named program, he may use the SAVE command to save the current program under a different filename, a procedure described in the following section.

### 8.2.1 Saving the Current Program under a Different Name

If for any reason the filename of the current program is not the one the user desires, he may save the program under a different name by issuing the SAVE command in the following form:

```
SAVE NEWNAM
```

This command saves the program currently in memory on the public structure under the name NEWNAM.BAS. When writing the file to any storage device, the SAVE command appends the .BAS extension by default.

The user may specify an extension other than the default .BAS by including a filename and his chosen extension in the SAVE command. The following example illustrates:

```
SAVE NEWNAM.E66
```

As a result of this command, the current program is saved on the public structure as NEWNAM.E66.

The SAVE command may be used with a complete file specification of the form

```
dev: [acct] filename.ext<prot>/option(s)
```

See Chapter 12 for a description of the complete file specification.

### 8.2.2 Using SAVE to Specify a Storage Device

The user may wish to save the current program, renamed or not, on a storage device other than the public structure. To store the program, under its current filename or a new filename, the user types a SAVE command in one of the following forms:

| | |
|---|---|
| SAVE dev: | which stores the program on the device specified as dev:, or |
| SAVE dev:NEWNAM | which stores the program as NEWNAM.BAS on the device specified as dev:. |

The following command, for example, saves, on DECtape unit 0, a copy of the program currently in memory. The program is saved under the filename ARCH.BAS.

```
SAVE DT0:ARCH
```

### 8.2.3 Using SAVE to Obtain Line Printer and Paper Tape Output

To obtain a line printer listing of the current program, the user types

```
SAVE LP:
```

To punch a paper tape of the current program on the high-speed punch, the user types

```
SAVE PP:
```

Because both devices are output-only and non-file structured, the user need not specify a filename.

> **NOTE**
>
> To punch a tape on the low-speed punch (on the ASR 33 terminal) the user may issue a LISTNH command, and, before pressing RETURN, turn the punch on line. This procedure is not recommended, however, because the word READY is punched at the end of the tape.
>
> Tapes punched on the low-speed punch should be read only through the low-speed reader.

## 8.3 CALLING AN EXISTING PROGRAM: THE OLD COMMAND

The OLD command allows the user to retrieve the source file of a previously saved BASIC-PLUS program; OLD causes any program currently in memory to be overwritten. This command is used to retrieve source programs only (.BAS files by default), since compiled programs (.BAC files) can be run but not changed. Thus, any program called with the OLD command can be edited by the user at the terminal.

BASIC-PLUS source programs can be given extensions other than the default .BAS.

To issue the OLD command, the user types

```
OLD
```

and presses the RETURN key. The system responds by printing the following request for the program's filename:

```
OLD FILE NAME --
```

The user then types the filename of the saved program. Or, the user may avoid the prompt OLD FILE NAME-- by typing the old filename on the same line as the command, as follows:

```
OLD TAXES
```

This command calls the saved file TAXES.BAS from the public structure. If the file is unavailable or protected against the user, an appropriate message is printed.

If the file has an extension other than .BAS, the user must specify that extension, as in the following example:

```
OLD TAXES.TL1
```

If the user does not respond to the OLD FILE NAME-- prompt with a filename, but presses RETURN instead, RSTS/E looks for the file NONAME (which the user or the system may have created; see Section 8.1.1.1). In the following example, the user retrieves the file NONAME:

```
OLD
OLD FILE NAME --

READY
```

As a result of this procedure, whatever was stored in the file NONAME.BAS on the public structure under the user's account is now in memory and available to the user.

The OLD command may be used with a complete file specification of the form

```
dev: [acct] filename.ext/option(s)
```

See Chapter 12 for a description of the complete file specification.

## 8.4 RUNNING AND COMPILING PROGRAMS: THE RUN AND COMPILE COMMANDS

### 8.4.1 Running a Program from the Public Structure: The RUN Command

The RUN command is used to execute any BASIC-PLUS source or compiled program. (Source programs are stored as typed by the user; compiled programs are described in Section 8.4.3.)

In order to identify and run the program that is currently in memory, the user types

```
RUN
```

This command not only runs (executes) the current program, but also prints, before execution, a program header consisting of the program's name and the current (system) date and time. If the user does not require this information, he should type the RUN No Header command:

```
RUNNH
```

which executes the current program, but does not print the header material. The RUNNH command runs only the program currently in memory; and filename or characters specified with this command are ignored.

To execute an "old" program – one not currently in memory – the user types the RUN command and the "old" program's filename, in the following form:

```
RUN PROG35
```

This command causes RSTS/E to search the public structure for the file PROG35.BAC or PROG35.BAS, and then to load it, to compile it if necessary, and to run it. Any current program is overwritten. This procedure, of course, assumes that the file can be found.

**NOTE**

Because blanks are not significant in BASIC-PLUS commands, the RUN filename command does not execute properly if the filename begins with NH. RUN NHTAX, for example, is the same as a RUNNH command. It will therefore run the program currently in memory. In the EXTEND mode of BASIC-PLUS, however, which does treat blanks as significant, RUN NHTAX is an invalid command. (See Section 9.2.1 for a description of EXTEND mode.)

If the source file PROG35.BAS and the compiled file PROG35.BAC both exist, RSTS/E loads and executes PROG35.BAC – because it is already compiled and uses less time. Note that because the program loaded is the compiled version, it can only be run – not edited or modified. If, however, the user wishes to execute PROG35.BAS, he may first retrieve it by issuing the OLD command, then execute it by issuing the RUN command; or, he may type the command RUN PROG35.BAS. After the program has been retrieved by OLD, it is currently in memory and may be edited or modified.

To continue with this example, assume that the compiled program PROG35.BAC does not exist. Only the source program PROG35.BAS exists. In this case, the command

```
RUN PROG35
```

loads, compiles, and executes PROG35.BAS, the source program.

The RUN command may be used with a complete file specification of the form

```
dev: [acct] filename.ext/option(s)
```

See Chapter 12 for a description of the complete file specification.

**8.4.2 Running Programs from Private or Specific Devices: The RUN dev: Command**
To run a program stored on a device other than the public structure, or stored on a particular device in the public structure, the user types RUN, the device specification, and the program's filename. The form is as follows:

```
RUN dev:FILNAM
```

The following command, for example, runs the file TRANS.BAC or TRANS.BAS, which resides on DECpack unit 1:

```
RUN DK1:TRANS
```

To read a source program from the high-speed paper tape reader and run that program, the user types the command

```
RUN PR:
```

Since the reader PR: is an input-only, non-file structured device, the user need not specify a filename.

If, on the other hand, the user does specify a filename with a non-file structured device in the RUN command, that filename is used as the current program name when the program is read into memory. In the following sequence, for example, a user issues a command to run a program from the high-speed reader, names the program SALE, and receives its output:

```
RUN PR:SALE
AVERAGE SALE FOR JULY:      77.26

READY
```

If, however, the user does not specify a filename, the program in memory will have no name. It is important to remember, in this case, that a simple SAVE – or any other command without a filename – cannot be applied to the program in memory unless that file has a name (see the RENAME command, Section 8.5). In the following sequence, a user runs a program from the high speed reader, but – unlike the user in the previous example – does not name the program in the RUN command. Thus, when this user attempts to save the program, he receives an error message:

```
RUN PR:
66 ELECTORAL VOTES STILL NEEDED TO NOMINATE

READY

SAVE
?ILLEGAL FILE NAME

READY
```

In order to save this program, the user must give it a filename in the SAVE command or name it by another method – the RENAME command, for example (see Section 8.5).

### 8.4.3 The COMPILE Command

**8.4.3.1 The Purpose of COMPILE** – Normally, RSTS/E accepts each line of a program as the user enters it; if a line is syntactically correct, RSTS/E then translates it into a form that is understood by the BASIC-PLUS system. This translation is called compiling of the program.

When the user edits the program, only the changed lines are compiled – that is, translated. And when the user gives the SAVE command, only the source version of the program (i.e., text that is typed in response to the LIST command) is stored in the .BAS file created. Thus, when the user gives the OLD command, the BASIC-PLUS system, in response, must first read the text of the saved program, and then must compile it – translate it – line by line again, just as it did when the user originally entered the program from the keyboard.

This process of compiling a saved program every time it is brought into memory may consume considerable system time. To avoid such repeated compilation, the user may issue the COMPILE command. COMPILE permits the user to save an image of his compiled program, rather than (or in addition to) the source text of the program. This compiled version, stored with the default extension .BAC, can be read from the disk with a minimum of system time.

**NOTE**

Compiled files have a minimum size requirement of 7 blocks. This size may be greater than necessary to actually store the compiled (or even the source) version of a short program. In such cases, the user should be aware that he is trading disk space for execution speed.

Because of the transformation that occurs when a program is compiled, a .BAC file can only be executed; it cannot be edited or modified. Therefore, a compiled program cannot be retrieved by the OLD command, whose function is to make a program available for editing. A compiled program must be called by the RUN command, which merely executes it.

**8.4.3.2 Using COMPILE** – If the program's current filename (i.e., the name printed in the header) is ACT01, then the command

```
COMPILE
```

saves the compiled program under filename ACT01.BAC on the public structure under the user's account. This command does not alter or replace ACT01.BAS.

If the user desires another name for the compiled program, he may specify it in a command of the following form:

```
COMPILE SCENE2
```

The preceding command creates, on the public structure, a compiled program named SCENE2.BAC.

The COMPILE command:

1. outputs compiled programs only to the disk structure,
2. appends the extension .BAC (unless otherwise specified) to the current or specified filename for storage in the user's public disk area, and
3. stores compiled programs with a default file protection of <124>; this default consists of the compiled file protection code <64> plus the system-assigned default code <60>. (Specifying another protection is discussed in the following paragraph.)

If the user wishes to add, to the compiled code <64>, a code other than the system default <60>, he must specify his choice of code in a COMPILE command of the following form:

```
COMPILE <40>
```

This command creates, on the public structure, a file with the current filename, the extension .BAC and the assigned protection <104>. This protection consists of the compiled code <64> and the user-specified code <40>.

At any time, a protection code can be altered by the BASIC-PLUS system command NAME AS (see Section 8.7), or by the renaming facility of the PIP system library program (see Chapter 17).

The COMPILE command may be used with a complete file specification of the form

```
dev: [acct] filename.ext<prot>/option(s)
```

See Chapter 12 for a description of the complete file specification.

**8.4.3.3 Compiling a Program on a Specific Disk Pack** – To compile the current program – under its current filename or a new filename – on a specific disk pack, the user types a COMPILE command in one of the following forms:

| | |
|---|---|
| COMPILE dev: | which compiles the program, under its current filename, on the disk specified as dev:, or |
| COMPILE dev:NEWNAM | which compiles the program as NEWNAM.BAC on the disk pack specified as dev:. |

Note that in either case the specified device must be a disk pack.

The following command, for example, compiles the program currently in memory on disk pack DK2:. The program is compiled as TELL.BAC.

```
COMPILE DK2:TELL.BAC
```

## 8.5 RENAMING THE CURRENT PROGRAM: THE RENAME COMMAND

The RENAME command allows the user to change the name of the program currently in memory. To issue this command, the user types RENAME, the new name for the program, and the RETURN key in the following form:

```
RENAME NEWNAM
```

As a result of this command, the "old" name of the program currently in memory is discarded; the current program is now known by the name NEWNAM. If the user now issues a SAVE command, NEWNAM.BAS is stored on the public structure.

## 8.6 REPLACING A SAVED PROGRAM: THE REPLACE COMMAND

The REPLACE command outputs the program currently in memory to the public structure or anywhere else, where it replaces a program with the same filename. REPLACE performs the same function as SAVE, but destroys the old program of the same name – if it exists – without notifying the user. The user may issue REPLACE alone or with a filename, in one of the following forms:

```
REPLACE

  or

REPLACE FILNAM
```

REPLACE appends the extension .BAS to the filename already associated with the program or specified in the command. If the user specifies a filename, REPLACE assigns that name to the new – or replacement – version of the program. In such a case, the current name of the program in memory is ignored.

The REPLACE command may be used with a complete file specification of the form

```
dev: [acct] filename.ext<prot>/option(s)
```

See Chapter 12 for a description of the complete file specification.

## 8.7 CHANGING A PROGRAM'S FILE SPECIFICATION: THE NAME AS STATEMENT

### 8.7.1 Using NAME AS to Rename a Program

NAME AS is a BASIC-PLUS statement that can be used in immediate mode to rename and/or assign a new protection code to a disk or DECtape file. The user types this statement in the following format:

```
NAME<string>AS<string>
```

The specified file, the first <string> indicated, is renamed as the second <string> indicated.

Should the file reside on a device other than the public disk structure, that device must be specified in the first string; it may also – at the user's option – be specified in the second string. The NAME AS command, however, does not copy a file from one device to another. Therefore, if a device name is used in the second <string>, it must be the same as that specified in the first <string>. NAME AS assumes no default filename extensions; if the file to be renamed has an extension, each of the two strings must include an extension.

The following two statements, for example, are equivalent:

```
NAME "DT0:OLD.BAS" AS "NEW.BAS"

                        and

NAME "DT0:OLD.BAS" AS "DT0:NEW.BAS"
```

The reader should note, however, that since the NAME AS statement cannot perform device transfers, there is no need to mention DT0: twice; that is, the device of the new filename need not be specified. (For information on device transfers, see PIP, Chapter 17.)

To change or create only the extension of a disk file on the public structure, the user types a statement of the following form:

```
NAME "ARC" AS "ARC.01"
```

This statement changes only the extension of the disk file ARC (with no extension) to .01.

The following statement, however, is not advised because of the absence of an extension in the second string:

```
NAME "FILE1.BAS" AS "FILE2"
```

As a result of this statement, the renamed file will have a null extension: FILE2.

### 8.7.2 Using NAME AS to Change a Protection Code

The user may change a program's file protection code by specifying the new code, within angle brackets < >, as part of the second string. If specified, this protection code becomes that of the renamed file. If the user does not specify a protection code, the old protection code is retained.

If the user wishes to keep the program's present filename, and change only its protection code, he may type a NAME AS statement in the following form:

```
NAME "YORK.BAS" AS "YORK.BAS <40>"
```

This statement changes only the protection code of the program YORK.BAS stored on the public structure.

Note that only privileged users can use NAME AS to assign a protection code higher than <63>.

## 8.8 FINDING A PROGRAM'S CURRENT AND MAXIMUM LENGTH: THE LENGTH COMMAND

The LENGTH command prints, at the user terminal, a message containing the current program's length, and, in parentheses, the maximum length that the system allows for the program. To issue this command, the user types

```
LENGTH
```

and presses the RETURN key. An example follows:

```
LENGTH
5 (8) K OF MEMORY USED
```

In this example, the user is informed that the current program is 5K words long, and that its maximum is 8K words. The present length of the program is reported to the next highest 1K increment. ("K" is a common abbreviation for 1024.)

## 8.9 LISTING DEVICE DIRECTORIES: THE CATALOG COMMAND

The CATALOG or CAT command allows the user to list all the files in a device directory, under his own account or that of another user, or in the system library.

To list his own public disk structure directory, the user types a simple CATALOG or CAT command, followed by a carriage return. In response, the system prints the directory at the user terminal. An example follows:

```
CAT
AVERAG.BAS     2      60    28-Oct-76 28-Oct-76 02:37 PM
PERCNT.BAS     2      60    28-Oct-76 28-Oct-76 02:37 PM
TERM   .DOC    11     60    28-Oct-76 28-Oct-76 02:37 PM
REPDLO.TXT     43     60    28-Oct-76 28-Oct-76 02:38 PM
PHONE  .DIR    43     60    28-Oct-76 28-Oct-76 02:39 PM
EXPENS.BAS     29     60    28-Oct-76 28-Oct-76 02:39 PM
CHOICE.BAS     2      60    28-Oct-76 28-Oct-76 02:40 PM
```

```
SOURCE.DAT        20        16        28-Oct-76 28-Oct-76 02:41 PM
BACKUP.CMD        1         60        28-Oct-76 28-Oct-76 02:42 PM
XX     .TST       1         60        28-Oct-76 28-Oct-76 02:42 PM
VISITS.LST        14        40        28-Oct-76 28-Oct-76 02:45 PM
TEMP24.TMP        0         60        28-Oct-76 28-Oct-76 03:01 PM

Ready
```

To obtain a catalog of files stored under another user's account number on the public structure, the user issues a command in this form:

```
CATALOG [100,102]
```

This command lists the public structure files owned by user account [100,102].

To obtain a listing of all files in the system library account, the user types

```
CATALOG $

     or

CAT $
```

In either command, $ indicates account [1,2], the system library.

To obtain a catalog of files on a device other than the public structure the user issues the command:

```
CATALOG dev:
```

where dev: is a device specification (a user account specification may be included in such a command). For example,

```
CAT DT1:
```

lists all files on DECtape unit 1 (no account numbers are associated with DECtape files).

To lists all files stored under a specific account on a non-disk device, the user types a command in this form:

```
CATALOG MT1:    [200,220]
```

This command lists all files stored under account [200,220] on magtape unit 1.

## 8.10 USING SCALED ARITHMETIC: THE SCALE COMMAND

The SCALE command implements and controls the scaled arithmetic features of BASIC-PLUS. This feature is used to overcome accumulated round-off and truncation errors in fractional computations performed on systems using the floating point format. The feature enables the system to maintain decimal accuracy of fractional computations to a given number of places determined by a scale factor. Scaled arithmetic is described in Section 6.8 of the *BASIC-PLUS Language Manual*.

Users on a system with the double-precision, four word floating-point format may issue the SCALE command to control the scale factor. An attempt to use the SCALE command on systems with the single precision floating-point format produces the ?MISSING SPECIAL FEATURE error message.

To specify the scale factor to be used, the user types the SCALE command with a decimal integer between 0 and 6.

For example,

```
SCALE 2

READY
```

The SCALE 2 command sets the current scale factor to 2. Subsequently, all programs compiled for that job have a scale factor of 2. If an invalid scale factor is typed with the command, the system prints the ?SYNTAX ERROR message.

**NOTE**

SCALE is solely a system command and cannot be executed as a BASIC-PLUS statement. Also, a program cannot refer to or modify the scale factor.

Typing a value of 0 with the SCALE command disables the scaled arithmetic feature. For example,

```
SCALE 0

READY
```

The SCALE 0 command sets the scale factor to 0. Programs compiled for that job subsequently treat all floating-point calculations in the standard fashion. When a user logs a job onto the system, the initial scale factor is 0.

To determine the current scale value, the user types the SCALE command without a value. For example,

```
SCALE

6

READY
```

The system prints the current value (6) and the READY message. If the program in memory has a value set other than the current value, the system prints two values. For example,

```
SCALE

6,2

READY
```

The first value (6) indicates the current value for the job, and the second value (2) is the one set for the program currently in memory. If the scaled arithmetic option is currently disabled, the system prints a zero as the first value.

When a user loads source statements into memory, whether by an OLD command, a RUN command to a .BAS file, or by entering statements after a NEW command, the system sets the scale factor for the program in memory to the current scale factor.

```
SCALE 4

READY

OLD TEST
```

```
READY

LISTNH
10        A$ = "##.#####"
20        X = .12345
30        PRINT USING A$,X
40        END

READY

RUNNH
  0.12340

READY
```

If the user executes a RUN command for the program in memory, the system performs floating-point calculations using the scale factor of the program currently in memory. Similarly, the system uses the scale factor of the program in memory when executing immediate mode statements.

After loading source statements into memory, the user cannot change the scale factor for the program in memory. This action prevents the user from possibly executing floating-point calculations for a program, parts of which the system assigned a different scale value.

An attempt to execute such a program or source statement causes the %SCALE FACTOR INTERLOCK warning message. For example,

```
SCALE
4

READY

OLD TEST

READY

SCALE 6

READY

RUNNH
%SCALE FACTOR INTERLOCK
  0.12340

READY
```

**NOTE**

The %SCALE FACTOR INTERLOCK message may not appear, since the system manager may have suppressed its printing. On such systems, the message does not appear but the operation of the scale factor remains unchanged.

The program executes using the scale factor of the program in memory (4) rather than the current scale factor (6).

To execute such a program with a changed scale value, the user may type a SAVE or REPLACE command followed by an OLD or RUN command for the related file. For example,

```
REPLACE TEST

READY

OLD TEST

READY

RUNNH
 0.12345

READY

SCALE
6

READY
```

The system consequently executes the program with the changed scale factor 6.

The following example illustrates the effect of different scale values for the sample program TEST.BAS, listed earlier in this section.

```
SCALE 2

READY

OLD TEST

READY

RUNNH
 0.12000

READY

SCALE 4

READY

OLD TEST

READY

RUNNH
 0.12340

READY
```

The system loads the source file using the current scale factor 2. When executed, the program generates results to 2 decimal places, and truncates the remaining places. If the user changes the current scale factor and loads the same program, the system executes the program with changed scale factor and truncates the remaining places.

If the user stores the compiled file and later runs it with a different scale value in effect, the following sequence takes place:

```
SCALE 6

READY

OLD TEST

READY

COMPILE TEST

READY

SCALE 4

READY

RUN TEST
  0.12345

READY

SCALE
4,6

READY
```

The system loads the program TEST and executes it with the scale value (6) set when the user compiled it. The system does not use the current value (4) but rather the value of the compiled program (6). This action occurs whether the program runs by a RUN command or by a CHAIN command.

If a compiled program is currently in memory, the user cannot change its scale factor and run that program. Since it is impossible to alter the scale factor of a compiled program except by compiling the program again, the system generates the %SCALE FACTOR INTERLOCK warning message (see NOTE earlier in this section). For example,

```
SCALE
4,6

READY

RUNNH
%SCALE FACTOR INTERLOCK
  0.12345

READY
```

The program executes with the scale factor of the compiled program (6).

# CHAPTER 9
# EDITING AND MODIFYING A PROGRAM

## 9.1 EDITING AND MODIFYING THE PROGRAM

### 9.1.1 Printing the Program: The LIST Command

The LIST command prints, at the user's terminal, a clean copy of all or part of the current program. In order to be printed by a LIST command, the program must be in memory. If the user wishes to print a source program that is not in memory, he must first issue an OLD command, then a LIST command as described in this section. LIST is especially useful during and after an editing session in which the user changes the current program.

To print a complete copy of the current program as it exists in memory, the user types

```
LIST
```

When the system prints the whole program, it lists source lines in line number sequence – regardless of the order in which the user entered them.

To print a single line, the user types LIST followed by the line number as follows:

```
LIST 100
```

This command prints a copy of line 100.

To print a specified section of the program, the user types LIST followed by the number of that section's first line, a hyphen, and the number of that section's last line:

```
LIST 100-200
```

This command lists all program lines from 100 to 200, inclusive. If 100 and 200 do not exist in the program, any lines within the range 100 to 200 are listed. This form of the command is especially useful with a video terminal, because it prevents long programs from scrolling off the screen.

One or more single lines or program sections may be specified in a single LIST command by separating the individual elements with commas. For example, the command

```
LIST 25,34,60-85,95,200-220
```

prints lines 25, 34, and 95 along with the program lines between (and including) 60 and 85 and between (and including) 200 and 220. Lines or program sections need not be indicated in sequential order in the LIST command; they are printed out in the order that the user requests.

The system, in response to a LIST command, prints a program header containing the program name and the current system date and time. If the user does not desire this information (and during a normal editing session, he may not), he may issue the LISTNH command to avoid printing of the header.

In executing any form of the LIST or LISTNH command, RSTS/E prints the ? character at the left of a line it considers to be in error; for example:

```
LISTNH
10    LET A,B=25
?20   PRINT A+B
          .
          .
          .
READY
```

The following is a summary of the forms of LIST and LISTNH:

| **LIST Command** | **Meaning** |
|---|---|
| LIST | List the entire user program as it currently exists. |
| LISTNH | Same as LIST, but omit program header. |
| LIST n | List line n. |
| LISTNH n | List line n without the program header. |
| LIST m,n,o | List lines m, n, and o. |
| LISTNH m,n,o | List lines m, n, and o without the program header. |
| LIST n1-n2 | List lines n1 to n2, inclusive. |
| LISTNH n1-n2 | List lines n1 to n2, inclusive, without the program header. |

Extensive examples of program listings appear in the *BASIC-PLUS Language Manual.*

**NOTE**

LIST sends output to the user terminal only. If a line printer is available, the command SAVE LP: is the fastest way to obtain a printed copy of the program (see Section 8.2.3).

### 9.1.2 Deleting Lines: The DELETE Command

The DELETE command deletes one or more lines from the current program; the line or lines to be deleted are specified in the command. The user should note, however, that typing a simple DELETE, without line numbers, deletes all lines from the program.

To delete one line from the program, the user types DELETE followed by that line's number in the following form:

```
DELETE 100
```

This command deletes line 100. For convenience, however, the user may type only the number of the line to be deleted and the RETURN key; for example,

```
15
```

is equivalent to DELETE 15 and deletes line 15.

To delete a section of the program, the user types a command of the following form:

```
DELETE 100-200
```

This command deletes all lines between and including lines 100 and 200. If 100 and/or 200 do not exist in the program, any lines within the range 100 to 200 are deleted.

If the user wishes to delete several lines or groups of lines, he may specify them in one DELETE command, separating the elements with commas as follows:

```
    DELETE 100-200, 255, 300-400, 470, 1000-1100, 475
```

This command deletes all lines from 100 to 200, line 255, all lines from 300 to 400, lines 470 and 475, and all lines from 1000 to 1100. Note that lines or sections need not be listed in sequential order in the DELETE command.

**9.1.2.1 Cautionary Notes About DELETE** – The user should remember that typing DELETE without a line number deletes all lines from the program.

Before deleting any line, the user should check for other lines containing references to the line he intends to delete – GOTO statements, for example. Obviously, such lines must be replaced or modified to reflect deletions of the lines that they reference. When the program is run, a reference to a missing line number will generate the error message ?STATEMENT NOT FOUND and halt the program's execution.

### 9.1.3 Simple Erasures: The RUBOUT Key and CTRL/U

**9.1.3.1 Erasing One Character at a Time: RUBOUT** – Typing the RUBOUT key (the DELETE key on some terminals) causes the last character typed to be erased. If typed in tape mode (see Section 5.8.1), the RUBOUT key is ignored.

If typed at a display (CRT) terminal, the RUBOUT key moves the cursor back one space and deletes the last character typed. No characters are echoed, as in the case of a hard-copy terminal.

If typed at a hard-copy terminal, the RUBOUT key causes the erased character or characters (if RUBOUT is repeatedly typed) to be echoed between backslashes. In the following example, the user has mistakenly typed LEF for LET:

```
    10    LEF  X=X*X
```

The user can correct this error by typing the RUBOUT key 7 times (to remove the F) and typing the remainder of the line correctly. If the user were to follow this procedure, the line would look like this on the terminal paper:

```
    10    LEF  X=X*X\X*X=X  F\T  X=X*X
```

To the system the line would appear as

```
    10    LET  X=X*X
```

In cases where the mistake is toward the beginning of a line, it may be easier to simply retype the entire line, as shown in the following example:

```
    10    LEF  X=X*X
    ?ILLEGAL VERB AT LINE 10

    READY

    10    LET  X=X*X
```

Once the second line (the corrected one) is entered to the system, the first line number 10 is deleted.

RUBOUT may be typed repeatedly on a display terminal, just as it may on a hard-copy terminal. The only differences in its effect are that characters are not echoed but are in fact erased, no backslashes appear, and the cursor is moved back one space for each character erased.

**9.1.3.2 Erasing One Line at a Time: CTRL/U** – The CTRL/U combination deletes the current input line. This combination is typed by holding down the CTRL key, typing U, and releasing both keys. CTRL/U is particularly useful when the user has typed a long line which he notices is incorrect. Rather than type RUBOUT repeatedly, the user may type CTRL/U to delete the entire line. This feature may be used when typing either commands or statements. CTRL/U deletes the entire physical line, as in the following example:

```
10 PPRINT "ALPHABET" ↑U
LISTNH

READY
```

In typing line 10, the user made a mistake (PPRINT for PRINT) and deleted the line with CTRL/U. Note that the line does not appear in the program listing caused by the LISTNH command.

In the next example, the user has typed the LINE FEED key to continue line 20 onto a second line. Thus, a physical line has been terminated.

```
20 LET M =
278. 12↑U

SYNTAX ERROR AT LINE 20

READY
```

The logical statement at line 20, however, is continued onto the following line and its deletion with a CTRL/U causes the statement at line 20 to be entered as

```
LISTNH
?20 LET M =

READY
```

which is syntactically incorrect. Had the last line been terminated with the RETURN key it would be entered to the system as

```
20 LET M =
278.12

READY
```

which would be accepted as equivalent to

```
20 LET M = 278.12
```

### 9.1.4 Removing a Program from a Storage Device: The UNSAVE Command and the KILL Statement

**9.1.4.1 The UNSAVE Command** – The UNSAVE command removes a .BAS or .BAC file from a storage device. To remove a file from the public disk structure (the default device), the user types a command in this form:

```
UNSAVE VOR45.BAC
```

This command removes the file VOR45.BAC from the public structure.

Unless a filename extension is specified in the UNSAVE command, the extension BAS is assumed. Thus, if the user issues the command

```
UNSAVE VOR45
```

The system attempts to remove the file VOR45.BAS.

To remove a file from a storage device other than the public disk structure, the user issues an UNSAVE in this form:

```
UNSAVE dev:filename.extension
```

where dev: is the device designation. For example, the command

```
UNSAVE DT1:FLIX
```

removes the file FLIX.BAS from DECtape unit 1, if the file can be found.

To remove a file with an extension other than .BAS, the user must specify the extension in a command of the following form:

```
UNSAVE FILE.001
```

The null extension cannot be used.

**9.1.4.2 The KILL Statement** – The KILL statement, placed within a program, deletes a specified file from a user's directory. This statement takes the following form:

```
<line number>    KILL<string>
```

The string can be a full file specification of the form

```
dev: [acct] filename.ext
```

If the specified file does not exist, the system prints the error message ?CAN'T FIND FILE OR ACCOUNT. If the specified device does not exist, the system prints the error message ?NOT A VALID DEVICE.

In order to delete a file with the KILL statement, the user must have write access to the file.

The KILL statement may also be used in immediate mode. Note that in the file specification no defaults are applied for filename or extension. The <string> may be either a string variable or a literal string enclosed in quotes.

### 9.1.5 Merging Programs: The APPEND Command

The APPEND command merges the contents of a previously saved BASIC-PLUS program into a BASIC-PLUS program currently in memory. To issue this command, the user types

```
APPEND
```

The system responds by printing the request

```
OLD FILE NAME --
```

The user then types the name of the saved program to be appended. This program is read into memory, and, depending upon the ordering of its source statement line numbers and those of the current program, the saved program's contents are merged into or appended to the current program. If both programs contain an identical line number, the line in memory is replaced by the appended program line.

The user may issue an APPEND without receiving the system's prompt by typing a command of this form:

```
APPEND RUTINE
```

This command merges the contents of the saved program RUTINE.BAS from the public structure into the program currently in memory. If the specified program is not available on the public structure or is protected against the user, an appropriate message is printed.

If the user does not specify a filename in response to the prompt OLD FILE NAME --, RSTS/E looks for the file NONAME.BAS, which could have been created by the user or by the system. (See Section 8.1.1.1 for a description of NONAME.BAS.) In the following example, the user, in response to the prompt, does not specify a filename, but merely presses the RETURN key after receiving the prompt:

```
APPEND
OLD FILE NAME --

READY
```

As a result of this procedure, the contents of the program NONAME.BAS on the public structure for the current user are merged into or appended to the current program. Thus, the contents of NONAME.BAS become available to the user as part of the current program.

In order to retain, under his account, the merged program that results from an APPEND, the user must issue a SAVE or a REPLACE command. If he uses SAVE, each of the component programs still exists separately under his account.

The APPEND command can retrieve only BASIC-PLUS source programs (.BAS files), because compiled programs (.BAC files) can be run but not changed. Any file called with APPEND is available to the user at the terminal.

The APPEND command may be used with a complete file specification of the form

```
dev:[acct]filename.ext<prot>/option(s)
```

See Chapter 12 for a description of the complete file specification.

### 9.1.6 Transferring Control Between Programs: The CHAIN Statement

CHAIN is a BASIC-PLUS statement that transfers control from one program to another. CHAIN may be used in immediate mode or as part of a program.

If a user program is too large to be loaded into memory and run in one operation, the user may segment it into two or more separate programs. In doing so, the user assigns each of these programs a unique name. To transfer control from one of these programs to another, the user issues the CHAIN statement.

In immediate mode, the form of the CHAIN statement is

```
CHAIN <string> <line number>
```

in which <string> is the file specification of the next program segment and <line number> is the line number in that program at which execution is to begin. If no line number is specified, execution begins with the lowest numbered line.

The following example is a CHAIN statement given in immediate mode:

```
CHAIN "PHASE2" 20
```

This statement executes the segmented program PHASE2, beginning with line 20. In its execution, CHAIN proceeds as follows: it first searches for the file PHASE2.BAC; if that search fails, it then searches for PHASE2.BAS. A device specification, project-programmer number, extension, and filename can be included in the file specification string.

Communication between various program elements can be achieved by means of a user's file or core common (see the discussion of system function calls in the *RSTS/E Programming Manual*).

When the CHAIN statement is executed, all currently open files are closed, the new program is loaded, and execution continues. CHAIN loads the designated program into memory and overwrites the current program. The CHAIN statement is similar to the RUN command, but has the additional capability of specifying the number of the line at which execution is to start.

Since each CHAIN statement closes all files, and causes a program load, the programmer should be careful to minimize the number of CHAIN statements that must be executed in the run of a program.

**NOTE**

It is recommended that the user explicitly close all open file channels by placing CLOSE statements in the program. The reason for this precaution is that the implicit closing feature of CHAIN may not allow the last write to be performed for virtual core and sequential files.

The <string> in the CHAIN command may be a file specification of the form

```
dev: [acct] filename.ext
```

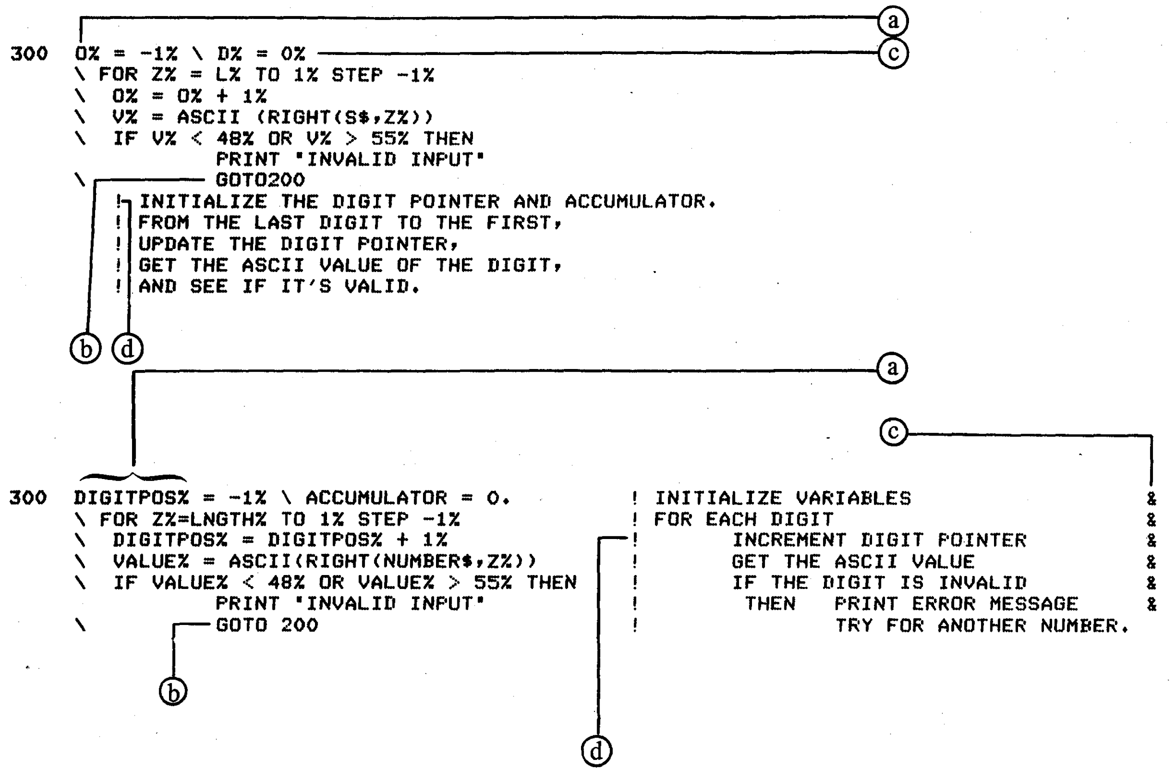
See Chapter 12 for a description of the complete file specification.

## 9.2 FORMATTING THE PROGRAM

### 9.2.1 Changing Statement Format Rules: The EXTEND/NO EXTEND Commands

**9.2.1.1 Description of EXTEND Format** – BASIC-PLUS offers two formats, EXTEND and NO EXTEND. EXTEND, when chosen by the user or when installed as a system default, allows expansion of statements and commands beyond the NO EXTEND format.

The following examples illustrate the differences between NO EXTEND and EXTEND format. The first example is a line from the BASIC-PLUS program OCTDEC.BAS, which appears in its entirety in Section 10.4, "An Example of Program Debugging." The second example shows this same line, 300, written in EXTEND mode. Points of difference in format are labeled in both examples and are described in the paragraphs that follow.



```
300  0% = -1% \ D% = 0%
     \ FOR Z% = L% TO 1% STEP -1%
     \  0% = 0% + 1%
     \  V% = ASCII (RIGHT(S$,Z%))
     \  IF V% < 48% OR V% > 55% THEN
                  PRINT "INVALID INPUT"
     \            GOTO200
        ! INITIALIZE THE DIGIT POINTER AND ACCUMULATOR.
        ! FROM THE LAST DIGIT TO THE FIRST,
        ! UPDATE THE DIGIT POINTER,
        ! GET THE ASCII VALUE OF THE DIGIT,
        ! AND SEE IF IT'S VALID.
```

```
300  DIGITPOS% = -1% \ ACCUMULATOR = 0.       ! INITIALIZE VARIABLES              &
     \ FOR Z%=LNGTH% TO 1% STEP -1%           ! FOR EACH DIGIT                    &
     \  DIGITPOS% = DIGITPOS% + 1%            !       INCREMENT DIGIT POINTER     &
     \  VALUE% = ASCII(RIGHT(NUMBER$,Z%))     !       GET THE ASCII VALUE         &
     \  IF VALUE% < 48% OR VALUE% > 55% THEN  !       IF THE DIGIT IS INVALID     &
                  PRINT "INVALID INPUT"       !        THEN   PRINT ERROR MESSAGE &
     \            GOTO 200                    !               TRY FOR ANOTHER NUMBER.
```

a. In EXTEND format, variable/function names can have up to 29 alphanumeric (and dot) characters in addition to the leading alphanumeric character, any FN prefix and/or $ or % suffix.

   In creating such names, the user should avoid mimicking existing BASIC-PLUS keywords such as statements and built-in function names (for example, IF, GOTO, LEN, etc.).

   NO EXTEND format allows only 1 alphabetic character or 1 alphabetic and 1 numeric character where EXTEND allows the 29 alphanumeric and dot characters.

b. In EXTEND format, spaces and tabs are significant; thus, the statement GOTO200 and the command ASSIGNDT1: are illegal, but GOTO 200 and ASSIGN DT1: are legal.

   NO EXTEND format ignores spaces and tabs; thus, GOTO200, GOTO 200, ASSIGNDT1:, and ASSIGN DT1: are all legal.

c. In EXTEND format, a BASIC-PLUS program line ending with

   &[<space/tab>sequence] <CR>

   or

   <LF>

   signals continuation of the logical line on the next physical line.

   NO EXTEND format uses a LINE FEED to signal continuation of the logical line on the next physical line.

d. In EXTEND format, a BASIC-PLUS program line ending with

   !<random text>& [<space/tab>sequence] <CR>

signals a comment on the same physical line, and continuation of the logical line on the next physical line.

NO EXTEND format requires the user to finish typing the logical line, then place the comment on the next physical line.

**9.2.1.2 Issuing the EXTEND/NO EXTEND Commands** – Issued in immediate mode, the EXTEND or NO EXTEND command changes both the user's current format and his default format to EXTEND or NO EXTEND, respectively. The default format is the one applied when the user issues the NEW or OLD command.

To issue one of the format commands, the user types

```
EXTEND
```

or

```
NO EXTEND
```

and follows either command by pressing the RETURN key.

Issued as a program statement, EXTEND or NO EXTEND changes only the user's current format to EXTEND or NO EXTEND respectively. The statement has the form

```
<line number>EXTEND
```

or

```
<line number>NO EXTEND
```

where line number is the number of the current line.

**9.2.2 Standard (NO EXTEND) Statement Formatting**
This section describes formatting features as used in NO EXTEND format, which is generally a system standard.

**9.2.2.1 Multiple Statements on a Single Line** – More than one statement can be written on a single line as long as each statement (except the last) is terminated with a colon or a backslash. (The backslash is preferred.) Thus only the first statement on a line can (and must) have a line number. For example,

```
10 INPUT A,B,C
```

is a single statement line, while

```
20  LET X=X+1.\ PRINT X,Y,Z\  IF Y=2. THEN GOTO 10
```

is a multiple-statement line containing three statements: a LET, a PRINT, and an IF-GOTO statement.

Any statement can be anywhere in a multiple-statement line except as noted in the discussion of the individual statements.

**9.2.2.2 A Single Statement on Multiple Lines** – A single statement can be continued on successive lines of the program. To indicate that a statement is to be continued, the user terminates the line with the LINE FEED key instead of the RETURN key. The LINE FEED performs a carriage return/line feed operation on the terminal, and the line to be continued does not contain a line number. For example,

```
10 LET W7=(W - X4*3.)*(Z - A/
(A - B) - 17.)
```

where the first line was terminated with the LINE FEED key is equivalent to

```
10 LET W7=(W - X4*3.)*(Z - A/(A - B) - 17.)
```

Note that the LINE FEED key does not cause a printed character to appear on the page.

The length of a multiple-line statement is limited to 255 continuation lines.

Where the LINE FEED key is used, it must occur between the elements of a BASIC statement. That is, a BASIC verb or the designation of a subscripted array element, for example, cannot be broken with a LINE FEED.

```
10 IF A1=0.
THEN GOTO 100
```

is acceptable where a LINE FEED follows 0., but:

```
10 IF A
1=0 THEN GOTO 100
?ILLEGAL CONDITIONAL CLAUSE
```

is not acceptable nor is:

```
10 IF A1=0. THEN GOTO 1
00
?MODIFIER ERROR AT LINE 10
```

and each illegal form generates an error message. A number of multi-word elements are processed as one word and cannot be broken by a LINE FEED. For example, AS FILE, FOR INPUT AS FILE, FOR OUTPUT AS FILE, GO TO, INPUT LINE, and ON ERROR GO TO are each treated by the system as one word.

9.2.2.3 **Spaces and Tabs for Readability** – Spaces should be used freely throughout the program to make statements easier to read. For example:

```
10 LET B = D*2. + 1.
```

instead of:

```
10LETB=D*2.+1.
```

or

```
10  L  ETB  =  D  *  2.  +  1.
```

The above statements are identical in effect.

TABS, like spaces, are used to make a program easy to read. An example follows:

```
20      IF A>B THEN
                IF B>C THEN
                        PRINT "A>B>C"
                ELSE    IF C>A THEN
                                PRINT "C>A>B"
                        ELSE    PRINT "A>C>B"
        ELSE    IF A>C THEN
                        PRINT "B>A>C"
                ELSE    IF B>C THEN
                                PRINT "B>C>A"
                        ELSE    PRINT "C>B>A"
```

# CHAPTER 10
# DEBUGGING A PROGRAM

The phase of program development during which the user is testing the program is called the debugging phase. BASIC-PLUS provides users with ways to debug their programs without having to run them over and over. These methods, described in this chapter, are immediate mode, the STOP statement, and the commands PRINT LINE, CONT, and CCONT (a privileged command).

## 10.1 EXECUTING STATEMENTS IN IMMEDIATE MODE

The immediate mode of BASIC-PLUS is so called because it enables one to enter a statement and to cause its immediate execution, without having to include that statement in a complete program. Immediate mode thus treats a BASIC-PLUS statement in much the same way as a command. In fact, as the reader may recall, some of the operations included in Chapters 8 and 9 as system commands – NAME AS and CHAIN, for instance – are actually BASIC-PLUS statements issued in immediate mode.

Immediate mode is especially useful for two general purposes: performing simple calculations which occur too infrequently to be programmed, and debugging programs – the subject of this chapter.

To execute a BASIC-PLUS statement in immediate mode, the user simply types that statement without a line number. BASIC-PLUS distinguishes between a programmed and an immediate statement solely by the presence or absence of a line number: numbered statements are stored; non-numbered statements are executed immediately on being entered. Thus, the statement

```
10              PRINT "STORE IT AWAY"
```

produces no effect at the user terminal upon entry. But a similar statement, entered without a line number, causes immediate output, as shown in the following example:

```
PRINT "DO IT NOW"
DO IT NOW

READY
```

The system prints the READY prompt to indicate its readiness for more input.

### 10.1.1 The Limitations of Immediate Mode

Before discussing the role of immediate mode in program debugging, it is useful to consider some of its limitations. Obviously, some BASIC-PLUS statements are too gregarious to "go it alone" in immediate mode. They need company: they must belong to a program and interact with its other statements in order to produce results. The following statements would be logically "stranded" in immediate mode, and would make no sense:

```
DEF
FNEND
DIM
DATA
FOR
NEXT
```

(Note, however, that FOR as a modifier will work in immediate mode.)

When the user gives any of these statements in immediate mode, the system prints the message ?ILLEGAL IN IMMEDIATE MODE.

Immediate mode does not permit multiple statements on a single line. Here, for example, the user attempts to enter two statements on one line, and receives an error message:

```
A=1.\   PRINT A
?ILLEGAL IN IMMEDIATE MODE

READY
```

## 10.2 USING IMMEDIATE MODE WITH STOP, CONT, CCONT, AND GOTO

### 10.2.1 The STOP Statement

Rather than repeatedly executing and altering a program, the user can facilitate debugging by strategically placing STOP statements throughout the program. Upon execution, each STOP statement causes a program halt, and a message of the form

```
STOP AT LINE 50
```

is printed. In this case, the STOP statement was located at line 50. After the STOP AT LINE message is printed, the user may give statements in immediate mode to examine and/or change data values. He may also add, delete, or modify lines. In addition, he may resume execution at another location by issuing a GOTO statement. When the user is ready to resume execution, he types CONT.

### 10.2.2 The CONT Command

After the user has performed debugging operations during a program halt caused by a STOP, he may continue program execution by issuing the CONT command as follows:

```
CONT
```

Execution then continues from the next statement after the STOP.

The user should note, however, that a syntax error or an illegal statement in immediate mode can prevent the CONT command from continuing program execution. When execution cannot be continued, the system prints the message

```
?CAN'T CONTINUE

READY
```

### 10.2.3 The CCONT Command

The CCONT command, available only to privileged users, performs the same actions as the CONT command but also detaches the job. Its special use lies in continuing a lengthy job that needs no further terminal interaction.

**NOTE**

Issued by a non-privileged user, CCONT produces the message ?ILLEGAL SYS( ) USAGE. To continue, the non-privileged user must issue the CONT command.

### 10.2.4 The GOTO Statement

The user can resume execution at a particular line by issuing the GOTO statement in immediate mode. Note that any such GOTO which causes transfer of control into or out of a FOR loop, function, or subroutine may cause unexpected results.

## 10.3 DEBUGGING WITH CTRL/C, PRINT LINE, AND CTRL/O

### 10.3.1 Halting and Checking Execution with CTRL/C and PRINT LINE

Another way to halt program execution is to type the CTRL/C combination (see Section 9.2.1.2). CTRL/C, however, gives the user less control over where the program halts than does the STOP statement.

When the program is halted by CTRL/C, the integer variable LINE contains the line number of the statement being executed at the time of the halt. Therefore, if the user types CTRL/C followed by PRINT LINE in immediate mode, the contents of LINE – i.e., the current line number – will be displayed at the terminal. The following example illustrates this procedure:

```
↑C

READY

PRINT LINE
300

READY
```

As when the program is halted by a STOP statement, the CONT command, CCONT command, or GOTO statement may be used to continue execution.

If a multi-statement line is being executed, the user has no way of knowing where in the line the program stopped. Some system programs are designed to trap CTRL/C to prevent their being interrupted within critical sequences.

### 10.3.2 Suppressing Output with CTRL/O

The CTRL/O combination suppresses output to the terminal without halting execution of the program; to continue output to the terminal, the user types another CTRL/O combination. Note that after a CTRL/O is issued, the program's output continues but is not printed at the terminal.

A program has no way of determining whether a CTRL/O has been typed at the terminal. There is, however, a system function call that overrides the ability of CTRL/O to suppress output.

### 10.3.3 Suppressing Output with CTRL/S and CTRL/Q

Also useful for temporary suppressing output on display (CRT) terminals only is the CTRL/S combination (the user holds down the CONTROL key and types S). To continue output to the terminal, the user types the CTRL/Q combination. These two features are usable only if the STALL characteristic is set on the terminal (see Section 20.1, TTYSET program).

## 10.4 AN EXAMPLE OF PROGRAM DEBUGGING

In the following example, the user has found that the program OCTDEC, written to convert octal numbers to decimal, returns incorrect results. (The octal number 23, for example, is converted to decimal number 56.) Employing STOP statements, CONT statements, and immediate mode, the user debugs the program. The step-by-step procedure appears after the listing of OCTDEC.

```
10      ! THIS IS AN OCTAL TO DECIMAL CONVERSION PROGRAM.

100     ON ERROR GOTO 900
        \ PRINT
        \ PRINT "OCTAL TO DECIMAL CONVERTER"
        \ PRINT "CONVERTS NUMBERS BETWEEN 0 AND 177777 (OCTAL) TO THEIR"
        \ PRINT " DECIMAL EQUIVALENTS"
                ! PRINT OUT THE INSTRUCTIONS AND HEADER.
```

```
200     INPUT 'OCTAL NUMBER';S$
        \ L% = LEN(S$)
        \ GOTO 600 IF L% = 0%
                ! INPUT A CHARACTER STRING,
                ! GET ITS LENGTH,
                ! AND CLOSE OUT IF ITS LENGTH IS 0.

250  STOP
300     O% = -1% \ D% = 0%
        \ FOR Z% = L% TO 1% STEP -1%
        \       O% = O% + 1%
        \       V% = ASCII (RIGHT(S$,Z%))
        \       IF V% < 48% OR V% > 55% THEN
                        PRINT 'INVALID INPUT'
        \               GOTO200
                ! INITIALIZE THE DIGIT POINTER AND ACCUMULATOR.
                ! FROM THE LAST DIGIT TO THE FIRST,
                ! UPDATE THE DIGIT POINTER,
                ! GET THE ASCII VALUE OF THE DIGIT,
                ! AND SEE IF ITS VALID.

350  STOP
400             V% = V% AND 7%
        \       D% = D% + (V% * INT(8% ^ O%))
        \ NEXT Z%
                ! CHANGE THE ASCII REPRESENTATION TO A NUMERIC
                ! REPRESENTATION.
                ! AND THE VALUE WILL ACCUMULATE IN D%.
                ! GO BACK AND DO NEXT DIGIT IF THERE IS ONE.

500     PRINT 'DECIMAL VALUE IS ';NUM1$(D%)
        \ GOTO 200
                ! PRINT OUT THE RESULT AND TRY ANOTHER.

600     GOTO 32767
900     IF ERL = 400 THEN
                PRINT 'NUMBER ';S$;' TOO BIG FOR CONVERSION'
        \       RESUME 200
910     PRINT ERR,ERL
        \ RESUME 32767
32767   END
```

1. The user suspects that the bug is in line 300 or 400; he therefore inserts a STOP statement before each of these lines, at lines 250 and 350. These STOP statements are underlined in the listing.

2. The user runs the program and inputs the octal number 23. The program stops at line 250 and so informs the user:

```
RUNNH

OCTAL TO DECIMAL CONVERTER
NUMBERS BETWEEN 0 AND 3641077
OCTAL NUMBER? 23
Stop at line 250

Ready
```

3. To check the value of variable L%, the user issues a PRINT statement in immediate mode:

```
PRINT L%
  2

Ready
```

4. Finding the value of L% correct (23 being 2 digits), the user issues a CONT statement to continue execution. The program stops on encountering the next STOP statement, and so informs the user:

```
CONT
Stop at line 350

Ready
```

5. To check the values of variables D, O%, V%, and Z%, the user issues another PRINT statement in immediate mode:

```
PRINT D, O%, V%, Z%
0              1              51              2

Ready
```

The user recognizes that the value of O%, printed as 1, should be 0 at this point in execution. (The program is dealing with the digit in position 0, and O% was initialized - 1.) The user finds the source of the error in the third physical line at 300, where O% (the letter variable) was mistyped as 0% (zero).

The user can now edit the program to correct the typographical error.

# PART IV

# SYSTEM LIBRARY PROGRAMS

# CHAPTER 11
# INTRODUCTION TO PART IV

Part IV of the User's Guide is largely devoted to the library of RSTS/E system programs. These programs perform diverse services for the RSTS/E user: simple functions such as printing a "sign" at the terminal that tells others it is in use, and more complex functions such as comparing and editing files, transferring data, and running batch jobs. In Part IV, these programs are organized by general purpose, as the chapter titles indicate. In this chapter, Table 11-1 is meant to provide the user with a guide to the library programs and the functions they perform.

Also contained in Part IV are two more introductory chapters – 12 and 13. Chapter 12, "FILE INFORMATION AND STANDARDS," sets forth an "anatomy" of the RSTS/E file specification – its parts, their meanings, their system-assigned defaults and the user's alternatives to those defaults. Since nearly all the programs in Part IV deal with files – the basic units of data in RSTS/E – the user should understand file specifications before reading about, and attempting to work with, most programs in the RSTS/E system library.

The next introductory chapter, 13, introduces the system library programs as a group by discussing their shared characteristics and features: for example, how some of them can be run by short CCL commands, and the ways in which they identify themselves and provide helpful information at the user's request.

Table 11-1 serves as a guide to the library programs, and to the RSTS/E file information found in Chapter 12. Table 11-2 lists the CCL (Concise Command Language) commands, which are brief ways to run some system programs. For more information on CCL commands, see Section 13.1.

**Table 11-1 Overview of Programs and File Information**

| Function | Program | Location |
|---|---|---|
| Account reporting | MONEY | 15.3 |
| Batch processing | BATCH | Chapter 22 |
| Comparing files | FILCOM | 16.3 |
| Creating, editing files | EDIT | 16.2 |
| Device, copying | COPY | 17.3 |
| Device, requesting | QUE | Chapter 21 |
| Device transfers | PIP, extended PIP | Chapter 17 |
| Directory, listing | DIRECT | 16.1 |
| Disk, floppy transfer | FLINT | Chapter 19 |
| Disk, mounting private | UMOUNT | 20.2 |
| Editing, creating files | EDIT | 16.2 |
| Floppy disk transfer | FLINT | Chapter 19 |
| Logging in | LOGIN | 14.1 |
| Logging out | LOGOUT | 14.2 |
| Message to manager | GRIPE | 15.2 |
| Message: "terminal in use" | INUSE | 15.3 |
| Preserving data off-line | BACKUP | Chapter 18 |
| Private disk mounting | UMOUNT | 20.2 |
| Quota report | QUOLST | 15.2 |
| Status report on system | SYSTAT | 15.1 |
| Terminal settings | TTYSET | 20.1 |
| "Terminal in use" sign | INUSE | 15.3 |

(continued on next page)

**Table 11-1 (Cont.) Overview of Programs and File Information**

| Elements of the file specification: | |
|---|---|
| **Element** | **Location** |
| device: | 12.1 |
| account (proj,prog) number | 12.2 |
| filename, extension | 12.3 |
| protection code | 12.4 |
| filespec options | 12.5 |

In Table 11-2, the shortest forms of the CCL commands appear outside square brackets. Any number of characters within square brackets may also be typed, in sequence. For example, the SYSTAT program may be run by typing any one of the following commands: SY, SYS, SYST, SYSTA, SYSTAT.[1]

**Table 11-2 CCL Commands that Run System Programs**

| Program | Command | Function of Command |
|---|---|---|
| DIRECT | DIR[ECTORY] | Lists standard directory. |
| | DIR[ECTORY]/option(s) | Lists directory according to option(s) specified. |
| EDIT | ED[IT] | Run EDIT program. |
| | ED[IT] OUT1,OUT2=IN1,IN2 | Sets up input and output files. |
| | ED[IT] filename | Sets up output file and .BAK (backup) file. |
| | CRE[ATE] filename | Sets up output file. |
| FLINT | FLI[NT] | Runs the FLINT dialogue. |
| UMOUNT | MOU[NT] device:pack id label/option(s) | Logically associates the specified disk pack with the specified identification label, and applies the specified option(s). |
| | DIS[MOUNT] device:pack id label | Logically dismounts the disk pack on the specified drive, and removes the specified pack identification label from system's table of logical names. |
| PIP | PIP | Runs the PIP program. |
| | PIP <command line> | Executes <command line> as a standard PIP command. |
| TTYSET | SET <command line> | Executes <command line> as a standard TTYSET command. |
| QUE | QU[EUE] <command line> | Executes <command line> as a standard QUE command. |
| SYSTAT | SY[STAT] | Prints standard system status report. |
| | SY[STAT]/option(s) | Prints system report according to option(s) specified. |

[1] Note that the CCL abbreviations in the table are those supplied by DIGITAL. A system manager can install additional CCL commands, and can alter any existing CCL commands or their abbreviations.

# CHAPTER 12
# FILE INFORMATION AND STANDARDS

A RSTS/E file specification contains some or all of the following information:

device: [proj,prog] filename.extension<protection>/option(s)

The elements of the file specification are discussed in the following sections. Here, Table 12-1 presents the default supplied by the system when the user does not specify an element of the file specification.

**Table 12-1 File Specification Elements and System Defaults**

| Element | Meaning | Default |
|---|---|---|
| device: | device designator | public disk structure (DF:,DK:,DP:,DB:,DM:, or SY:) |
| [proj,prog] or [account] | project-programmer or account number | the current user's account number |
| filename.extension | title of file.type of file | defaults depend upon current program and procedure. A file created at BASIC-PLUS command level, however, has the default name NONAME.BAS. |
| <protection> | protection code | on most systems, <60> |
| /option | cluster size, data mode, and/or total size of file | defaults for options, of which there are 4, depend on the system |

## 12.1 device: THE DEVICE DESIGNATOR

If the user does not specify a device designator, the system supplies the public structure by default. For non-file structured devices, (paper tape, line printer, etc.), only the device designator need be specified; the system ignores any filename, extension, account number, or protection code that the user specifies.

A device designator, or device specification, consists of 2 alphabetic characters optionally followed by a decimal unit number, and always terminated by a colon (:). The alphabetic characters are generally an abbreviation of the device's generic name (DT for DECtape, DK for disk, MT for magtape, etc.), and the unit number uniquely identifies the device itself or the drive on which it is currently mounted.

Table 12-2 lists and explains the RSTS/E device designators.

If a user attempts to specify a device or type of device not available on the system, the error message ?NOT A VALID DEVICE is printed.

### 12.1.1 Logical Device Names

A device may be identified either by a physical device name – i.e., a device designator – or by a logical device name; logical device names are associated with devices by the ASSIGN command (see Section 5.4) or by the system manager. A logical device name consists of 1 to 6 characters terminated by a colon. Examples of logical device names

**Table 12-2 RSTS/E Device Designators**

| Designator | Device |
|---|---|
| DF:, DK:, DP:, DB:, DM:, DS:, or SY: | RSTS/E public disk structure as a whole |
| SY0: | System disk (the unit which was bootstrapped) |
| DF0: | RF11 disk (all platters) |
| DK0: to DK7: | RK11/RK05 disk cartridge units 0 through 7 |
| DP0: to DP7: | RP11/RP02/RP03 disk pack units 0 through 7 |
| DB0: to DB7: | RH11/RP04/RP05/RP06 disk pack units 0 through 7 |
| DM0: to DM7: | RK611/RK06 disk cartridge |
| DS0: to DS7: | RH11/RS03/RS04 fixed head disk units 0 through 7 |
| PR: | High-speed paper tape reader |
| PP: | High-speed paper tape punch |
| CR: | CR11 punched or CM11 mark sense card reader |
| CD: | CD11 punched card reader |
| MT0: to MT7: | TM11/TU10 or TS03 magtape units 0 through 7 |
| MM0: to MM7: | TM02/TU16 or TU45 magtape units 0 through 7 |
| LP0: to LP7: | Line printer units 0 through 7 |
| DT0: to DT7: | TC11/TU56 DECtape units 0 through 7 |
| KB: | Current user terminal |
| KBn: | Terminal n in the system |
| TTn: | Terminal n in the system (synonym for KBn:) |
| TI: | Current terminal (synonym for KB:, the terminal that initiated the job) |
| DX0: to DX7: | Floppy disk units 0 to 7 |

**NOTE**

The user can reference LPn:, DTn:, DXn:, KBn:, MMn: and MTn: where n is between 0 and the maximum number of such units on the system. LP:, DT:, DX:, MM: and MT: are each the same as specifying unit 0 of the related device.

On systems which do not have TU10 or TS03 magtape units, the designator MT: is a synonym for MM:. On systems which have the CD11 card reader, the designator CR: is a synonym for CD:.

are DTA:, DTFACT:, MTMINE:, MYPACK:, DK5:. A project-programmer (account) number may be associated with a system-wide logical name.

**NOTE**

When a user specifies a device name, physical or logical, the system seeks the device by the following procedure: 1) the system determines if the device name is specific to one job; 2) if it is not, the system determines if it is a system-wide logical name; 3) if the device name is neither job-local nor system-wide, the system determines if it is a physical device designator.

## 12.2 [proj,prog] THE PROJECT-PROGRAMMER OR ACCOUNT NUMBER

If the user does not specify a project-programmer code, the system assumes his own number by default; that is, the owner of the file is assumed to be the current user. This code is meaningful only for disk and magtape files; it has no significance for files on DECtape or on non-file structured devices. The [proj,prog] code consists of two decimal numbers between 0 and 254, separated by a comma, and enclosed within square brackets [ ] or parentheses ( ). These numbers have the following meanings:

| | |
|---|---|
| proj | represents a project group consisting of users with a common task or interest. |
| prog | represents an individual user in that project group. |

RSTS/E uses project-programmer codes for two general purposes: 1) to identify files according to their owners, and 2) to apply the files' protection codes to individual users and to groups of users (see Section 12.4, "PROTECTION CODE").

### 12.2.1 Special Account Characters

In any location where a project-programmer code is valid, the user may specify one of the special non-alphanumeric characters listed below. Each of these characters designates an account. The $, for example, designates the system library account [1,2]. Thus, a file specification containing a $ denotes a file stored in that account. The usual application of $ is in calling a system library program – the command RUN $PIP, for example.

The special account characters and the accounts they designate are as follows:

| Character | Account |
|---|---|
| $ (CHR$(36)) | [1,2], system library account |
| ! (CHR$(33)) | [1,3], determined by manager |
| % (CHR$(37)) | [1,4], determined by manager |
| & (CHR$(38)) | [1,5], determined by manager |
| # (CHR$(35)) | [proj,0] |
| @ (CHR$(64)) | assignable account |

The # character allows each project on the system to have its own library of files common to all members of that project.

Assignment and use of the @ character are explained in Section 5.7. If the user specifies the @ character without having previously assigned it to an account, ?ILLEGAL FILE NAME error is printed.

## 12.3 filename.extension THE FILENAME AND EXTENSION

For file structured devices, each file is assigned a filename and extension.

The filename is a string of one to six alphanumeric characters, without embedded nulls, tabs, or spaces. It is the only element of a file specification without a delimiting mark of punctuation. The filename may also include – or consist of – the "wild card" character ?. Or, it may consist of only the "wild card" character * (see the following section, "Wild Card Specifications").

The filename extension is a string of one to three alphanumeric characters without embedded nulls, tabs, or spaces preceded by a dot (.). Usually, the extension denotes the file's type: .BAS, for example, indicates a BASIC-PLUS source file; .CTL, a program control file; .TMP, a temporary BASIC-PLUS file (see Table 12-3 for common RSTS/E extensions and their meanings). An extension may include, or consist of, the "wild card" character ?. Or, it may consist of only the "wild card" character * (see the following section, "Wild Card Specifications").

A null or blank extension is permitted, in which case the dot and filename extension field are omitted from the file designation.

**Table 12-3 RSTS/E Filename Extensions**

| Extension | Meaning |
|---|---|
| .B2S | BASIC-PLUS II source file |
| .BAC | executable BASIC-PLUS program |
| .BAS | BASIC-PLUS source file |
| .BAK | BAcKup file created by EDIT program |
| .CBL | COBOL source file |
| .CMD | indirect CoMmanD file for running a system program |
| .CTL | BATCH ConTroL file |
| .DOC | RUNOFF output file |
| .FOR | FORTRAN source file |
| .LOG | BATCH output LOG file |
| .LST | LiSTing file created by a system program |
| .OBJ | compiled COBOL, FORTRAN, or BASIC-PLUS II program |
| .ODL | Overlay Description Language input file |
| .RNO | RUNOFF source file |
| .SAV | compiled and linked FORTRAN program |
| .TMP | TeMPorary file created by a system program |
| .TSK | Executable COBOL or BASIC-PLUS II program |
| .TXT | ASCII TeXT file |

### 12.3.1 Wild Card Specifications

Many of the RSTS/E system library programs, when requiring a file specification, allow the user to specify the wild card characters * (asterisk) and ? (question mark). The * character replaces an entire field, while the ? character replaces a character within a field.

**12.3.1.1 The * Wild Card** – The * (asterisk), replacing a filename, extension, or both (i.e., one * for each), denotes all filenames, all extensions, or all filenames with any extensions. The following examples illustrate:

| | |
|---|---|
| FILE.* | denotes all files with filename FILE and any extension; for example, FILE.DAT, FILE.TXT, FILE.8, FILE.9B. |
| *.EXT | denotes all files with EXT extensions; for example, DATA.EXT, MYFILE.EXT, FB10.EXT, DD.EXT. |
| * . * | denotes all files; for example, all the files listed as examples above. |

**12.3.1.2 The ? Wild Card** – The ? (question mark), in any position of either the filename or extension, denotes any alphanumeric character appearing in that position. The following examples illustrate:

| | |
|---|---|
| FILE.EX? | denotes all files with filename FILE and an extension consisting of EX, or of EX and any other alphanumeric character; for example, FILE.EX1, FILE.EX2, FILE.EXF, FILE.EXE, FILE.EX. |
| FILE??.EXT | denotes all files with extension .EXT and a filename consisting of FILE and any other two alphanumerics, including trailing blanks; for example, FILE01.EXT, FILE54.EXT, FILE3B.EXT, FILE3.EXT. |
| FILE??.E?? | denotes all files with a filename consisting of FILE and any other two alphanumerics, and an extension consisting of E and any other two alphanumerics, (trailing blanks are included); for example, FILE54.EXT, FILE01.ERA, FILEJQ.E91, FILE91.EJQ. |
| FI?.EXT | denotes all files with extension .EXT and a filename consisting of FI and any other alphanumeric (trailing blanks are included); for example, FI1.EXT, FIL.EXT, FI7.EXT, FIG.EXT. |

**12.3.1.3 The * and ? Wild Cards Combined** – The * and ? wild cards may be intermixed in a file specification. The following examples show such mixtures and their meanings:

| | |
|---|---|
| FILE??.* | denotes all files with any extension and a filename consisting of FILE and any two alphanumerics; for example, FILE60.DAT, FILE75.DAT, FILEZX.TXT, FILES5.B, FILEB6.AM. |
| *.EX? | denotes all files with an extension consisting of EX and any other alphanumeric; for example, MYFILE.EXT, YRFILE.EXT, MCR.EXE. |

## 12.4 <protection> PROTECTION CODE

The protection code is a string of one to three decimal digits enclosed by angle brackets < >. This string determines the file's degree of protection on two levels: the actions – reading, writing, and deleting – against which it is protected, and the user or class of users against whom it is protected. There are three such user classes, which the system recognizes by project-programmer numbers as follows:

1. The individual user (owner)
   who is recognized by his programmer number: [200,25].
2. The user's project group
   which is recognized by the user's project number: [200,25],[200,57],[200,70].
3. All other users on the system
   who are recognized by the existence of valid project-programmer numbers: [225,60], [250,35],[254,10]

Thus, two variables – read/write privileges and class of user – determine protection. Degrees of protection for data files are enforced by the following individual codes; typically, a file's total protection code is the sum of the desired combination of individual codes. These individual codes and their meanings are listed in Table 12-4.

In accordance with the codes in Table 12-4, therefore, a data file with protection <60> – the usual system default – is protected against reading, writing, and deleting by all users except its owner: <60>= 4+8+16+32.

## 12.5 /option FILE SPECIFICATION OPTION

A file specification option may be included as the final element of the specification string. Three such options are possible: /FILESIZE, /CLUSTERSIZE, /MODE, and /RONLY. These options specify, respectively, the disk size – in blocks – to which the file is pre-extended, the minimum number of contiguous disk blocks forming a cluster, and the read/write mode in which the file's data is passed to the device driver. Note that /FILESIZE and /CLUSTERSIZE are primarily used for disk files.

**Table 12-4 File Protection Codes**

| Code | Meaning |
|---|---|
| 1 | read protection against owner |
| 2 | write protection against owner |
| 4 | read protection against owner's project group |
| 8 | write protection against owner's project group |
| 16 | read protection against all others who do not have owner's project number |
| 32 | write protection against all others who do not have owner's project number |
| 64 | executable program: can be run only |
| | Individual codes added to the compiled protection <64> have meanings different from those of the data file protection codes above. These compiled codes follow: |
| 1 | execute protection against owner |
| 2 | read and write protection against owner |
| 4 | execute protection against owner's project group |
| 8 | read and write protection against owner's project group |
| 16 | execute protection against all others who do not have owner's project number |
| 32 | read and write protection against all others who do not have owner's project number |
| 128 | program with temporary privileges (normally occurs only when file's protection includes <64>). |

If a file specification option is in a position other than the final one, is missing a colon, or is not a valid form, the system prints the error message ?ILLEGAL SWITCH USAGE. For example, either of the following specifications will produce the switch usage error:

```
ABC/SI:100 [1,2]
```

or

```
ABC/SIQ
```

If an argument is missing, contains an illegal character, or is greater than 65535, the system generates the ?ILLEGAL NUMBER error.

### 12.5.1 /FILESIZE Option

The /FILESIZE option allows the creation of a disk file of the specified size, in blocks, before any read/write operations are performed. Thus, /FILESIZE reserves space on the disk for data to be placed in the file. This option consists of 1) a slash (/), 2) any one of the unique abbreviations shown in the list that follows, 3) a colon (:), 4) an optional pound sign (#) for octal conversion of the argument n, 5) the argument n, a decimal number between 0 and 32767 inclusive, which indicates the number of blocks in the pre-extended file, and 6) an optional trailing decimal point (.) to ensure that n is interpreted as a decimal number.

The /FILESIZE option may be minimally abbreviated to /FI or to /SI. The following list shows its valid forms:

```
/FI: [#] n [.]
/FIL: [#] n [.]
/FILE: [#] n [.]
  .
  .
/FILESIZE: [#] n [.]
```

or

```
/SI: [#] n [.]
  .
  .
/SIZE: [#] n [.]
```

### 12.5.2 /CLUSTERSIZE Option

The /CLUSTERSIZE option establishes the minimum cluster size for a disk file; a cluster is a number of contiguous blocks taken together as a unit. The /CLUSTERSIZE option is especially useful for large files; specifying a large cluster size speeds up random access to the data and prevents such files from crowding or filling the user's directory, whose own size is limited. RSTS/E permits cluster sizes of 1, 2, 4, 8, 16, 32, 64, 128, or 256 blocks.

The /CLUSTERSIZE option consists of 1) a slash (/), 2) CLUSTERSIZE or a minimum abbreviation of CL, 3) a colon (:), 4) an optional minus sign (-) to specify a negative cluster size (explained in the next paragraph, 5) an optional pound sign (#) for octal interpretation of the argument n, 6) the argument n specifying the cluster size in blocks, and 7) a decimal point (.) to ensure decimal interpretation of n. The following list shows the valid forms of the option:

```
/CL: [ - ] [#] n [.]
/CLU: [ - ] [#] n [.]
/CLUS: [ - ] [#] n [.]
  .
  .
  .
/CLUSTERSIZE: [ - ] [#] n [.]
```

Specifying a negative cluster size is a way to avoid certain errors associated with disk devices. A negative cluster size tells the system to use the absolute value of the cluster size, if the device on which the file is created allows that value. If the ?ILLEGAL CLUSTER SIZE error is encountered because the absolute value is less than the cluster size at which the file is to be created, the system will use the device cluster size rather than return the error. For example, a user is accustomed to creating a file with a cluster size of 2 on an RK05 disk cartridge, which is a system scratch disk. The scratch disk, however, temporarily changes to an RP06 disk pack, which has a device cluster size of 8. The user's customary cluster size of 2, therefore, would be illegal on the RP06. But by specifying the negative cluster size of -2, the user guarantees that the file creation will not fail because of the RP06 disk's cluster size restriction.

**12.5.3 /MODE and /RONLY Options**

The /MODE option enables the passing of up to 15 (decimal) bits of information to the device driver at file open time. The meaning of these bits (if any) is device dependent, and determines the read/write mode for data transfer. For explanations of the bits, see the *RSTS/E Programming Manual.*

The /MODE option consists of 1) a slash (/), 2) MODE or a minimum abbreviation of MO, 3) a colon (:), 4) an optional pound sign (#) for octal interpretation of the argument n, 5) the argument n specifying a mode setting between 0 and 32767 (decimal) inclusive, and 6) an optional decimal point (.) to ensure decimal interpretation of n. The following list shows the valid forms of the option:

```
/MO:[#]n[.]
/MOD:[#]n[.]
/MODE:[#]n[.]
```

The /RONLY option enables setting of the read only MODE value for a disk file. The /RONLY option consists of 1) a slash (/), and 2) RONLY or a minimum abbreviation of RO. The following list shows the valid forms of the option:

```
/RO
/RON
/RONL
/RONLY
```

# CHAPTER 13
# PROGRAM INFORMATION AND CHARACTERISTICS

## 13.1 THE CONCISE COMMAND LANGUAGE (CCL)

A RSTS/E user is able to run some of the system library programs by typing a unique system command called a Concise Command Language (CCL) command. The number of programs that can be run by CCL commands is generally decided by the system manager, although Digital supplies CCL commands with some of the library programs. These standard CCL commands are listed in Chapter 11, the introduction to Part IV. Specific descriptions of Digital-supplied CCL commands are found in the chapters describing their associated programs.

The chief advantage of CCL commands is that they allow the user to call a system program with one brief command (by typing PIP, for example, instead of RUN $PIP), and to give that program a specific command to execute: QUE MYFILE.DAT, for example, calls the QUE program and also tells it to print MYFILE.DAT at the line-printer.

### 13.1.1 Cautionary Notes on Typing CCL Commands

**13.1.1.1 Embedded Spaces in CCL Commands** – The user should note that the CCL translator, unlike the BASIC-PLUS translator, interprets spaces embedded in strings as significant. Thus, if a user attempting to run the EDIT program types E DIT, he will not succeed in running the program and instead will receive an error message.

**13.1.1.2 Mistyping a BASIC-PLUS Command as a CCL Command** – When the user enters a line at BASIC-PLUS command level, RSTS/E first checks for a BASIC-PLUS line number. If RSTS/E finds a line number, it compiles the line as a program statement. But if it finds no line number, it sends the line to the CCL translator; if the CCL translator recognizes the line as a valid CCL command, it causes the appropriate program to be run, thus destroying the current contents of the user's area in memory. If, on the other hand, the CCL translator does not recognize the line as a valid CCL command, it sends the line back to the BASIC-PLUS system, which then attempts to execute the line as a BASIC-PLUS system command or immediate mode statement.

Thus, the user, in issuing a BASIC-PLUS system command or immediate mode statement, should be careful not to mistype that command or statement so that it mimics a valid CCL command: if it does, the user's area in memory – i.e., the current program – will be destroyed and replaced by a system library program.

It is unlikely that a BASIC-PLUS statement or command would be mistyped as any of the DIGITAL-supplied CCL commands. Some systems, however, may have CCL commands of their own that do resemble BASIC-PLUS statements or commands. Therefore, the user is strongly advised to become familiar with the Concise Command Language as it exists on his system.

## 13.2 OBTAINING HELP FILES FOR SYSTEM PROGRAMS

Many of the library programs offer the user assistance in the form of help files; these files contain information on running the program and using its commands and/or options, and may be printed at the user's terminal. Generally, the simple, one-command programs such as INUSE, GRIPE, MONEY, and QUOLST do not provide help files.

The method of requesting a help file depends upon the program. Some programs, for example, allow the user to print their help files without first running the program; for these programs, the user may type HELP followed by the program's name. Other programs must first be run before the user may request their help files – generally by typing the /HE option. And some programs allow both forms of request. The user should refer to individual program descriptions in Part IV.

## 13.3 VERSION IDENTIFICATION

Most system library programs, on being called by the RUN $ command or by a CCL command that merely calls the program, print a program header before the prompt. This header contains the program name and the RSTS/E version number, along with other information such as the system number and the current job number.

The following output, for example, is PIP's response to either the RUN $PIP command or the CCL command PIP:

```
PIP     V06B-03     RSTS V06B-02     TIMESHARING
#
```

For other examples of program headers, see the individual descriptions in Part IV.

## 13.4 INDIRECT COMMAND FILES

Two system programs, BACKUP and extended PIP, may be run by indirect command files. An indirect command file is created by the user and contains commands which the program executes as it reads the file. Indirect command files are useful when a sequence of operations is to be performed repeatedly; once the user has placed the commands for these operations in a file, he need not issue them one by one when he wishes to perform the operations.

Methods of creating and running indirect command files vary according to program. BACKUP, which runs by an ordered dialogue, involves methods different from those of PIP, which does not. See the individual program descriptions in Part IV.

# CHAPTER 14
# JOB CONTROL PROGRAMS

## 14.1 ENTERING THE SYSTEM: THE LOGIN PROGRAM

The LOGIN system program runs from either a logged in or a logged out terminal. It activates a job at a terminal, attaches a detached job[1] to a terminal, or runs designated system programs from a logged out terminal.

### 14.1.1 Running LOGIN from a Logged Out Terminal

As described in Chapter 2, the LOGIN system program runs when either HELLO, LOG, LOGIN, ATTACH, ATT or I is typed at a terminal connected to the RSTS/E system. This section describes more fully the actions which occur when LOGIN runs at a logged out terminal.

When a terminal is connected to the RSTS/E system by a dial up connection, the automatic answering signal causes the system monitor to insert an I in the input buffer for that terminal. This action simulates an I being typed at a terminal directly connected to the system. The description of the resultant actions in this section applies to the cases of a terminal directly connected to the system and of a terminal connected by a dial up line.

When a user enters typed characters to the system from a terminal directly connected to but not logged into the system, the monitor runs the LOGIN system program which checks the characters for a valid command. If only the RETURN key is typed, the system takes no action. A user entering either SYS or SET commands causes LOGIN to chain to the SYSTAT or TTYSET system programs, respectively. The system manager can alter the LOGIN program to run other programs in the same manner.

If the user types HELLO, LOGIN, LOG, ATTACH, ATT, or I, LOGIN prints the system identification line as in the following sample printout.

```
HELLO

RSTS V06B-02 Timesharing  Job 16  KB2  02-Nov-76  11:07 AM
#1,210
```

The line contains the system name and version number, the local installation name, the job number activated, the keyboard number of the terminal, and the current system date and time. The pound sign (#) character printed by LOGIN on the following line requests the user to type his account number.

The user types the project and programmer numbers separated by either a comma or a slash and terminated with the RETURN key. (Typing the slash as a separator inhibits printing of any system notice messages.) The user can specify the project and programmer numbers on the same line as the HELLO, LOGIN, LOG or I commands as shown in the following sample dialogue.

```
HELLO 1,210
Password:
```

When an account number is included in the command, LOGIN does not print the # character but immediately prompts the user to enter the password. LOGIN disables echo printing at the terminal when the password is typed.

[1] A job becomes detached because the connection of a dial up line is broken or a privileged job executed the SYS system function to detach the job from the terminal.

If either the account does not exist or the password does not match, LOGIN prints the INVALID ENTRY – TRY AGAIN message and the # prompting character. The user can try the sequence to a maximum of five times. LOGIN allows the user 30 seconds in which to type an entry. After the fifth invalid entry, LOGIN prints the ACCESS DENIED message and frees the job for other usage.

A valid entry causes LOGIN to check for any other jobs which may be running on the system under the same account number. If other jobs are running and none are detached, LOGIN reports how many such jobs by printing a message similar to the following sample and prints the system notice messages.

```
1 other user is logged in under this account
```

If any jobs are running detached under the current account, LOGIN instead reports the number of each such job and requests the user to type the number of the job to be attached to the terminal. The following sample printout shows the procedure.

```
Job 16 is detached under this account
Job number to attach to? 16
Attaching to Job 16

Ready
```

To attach a job to the terminal, simply type its number in response to the query. LOGIN prints the ATTACHING TO message and attempts to attach the specified job to the current terminal. When the job is attached, the current job is freed for other usage and the attached job runs at the terminal.

To continue running the current job, the user simply types the RETURN key in response to the query. LOGIN subsequently prints the message concerning other jobs running under the same account and prints the system notice messages, if any.

```
Job 16 is detached under this account
Job number to attach to?
2 other users are logged in under this account

Ready
```

System notices convey to the user information which the system manager places in the file NOTICE.TXT in the system library. If the file does not exist, LOGIN proceeds. LOGIN prints the READY message and exits to the system monitor which clears the LOGIN program from memory.

The complete sequence to log a job into the system when other jobs are running detached is shown below.

```
HELLO 1/210
Password:
Job 16 is detached under this account
Job number to attach to?
2 other users are logged in under this account

Ready
```

The complete sequence to attach another job to the terminal when logging into the system is shown in the following sample printout.

```
HELLO

RSTS V06B-02 Timesharing  Job 11  KB2  02-Nov-76  11:31 AM
#1/210
Password:
Job 16 is detached under this account
Job number to attach to? 16
Attaching to Job 16

Ready
```

To attach a job to a terminal when the job number is known, the user can type the ATTACH or ATT command as follows:

```
ATTACH

RSTS V06B-02 Timesharing  Job 13  KB2  02-Nov-76  12:01 PM
Job number to attach to? 34
Job not detached - access denied

Bye
```

LOGIN runs and prints the system identification line and, on the next line, the JOB NUMBER TO ATTACH TO query. The user must type the number of the detached job. If the job is not detached or does not exist, LOGIN prints an appropriate message followed by ACCESS DENIED. The user must type another command to log into the system.

If the job is detached, LOGIN prompts the user for the password of the account under which the detached job is running. After the user enters the password, LOGIN prints the ATTACHING TO JOB x message and attempts to attach the specified job to the terminal. An incorrect password causes LOGIN to print the FAILURE TO ATTACH TO JOB x message and to terminate as shown in the following sample printout.

```
ATTACH

RSTS V06B-02 Timesharing  Job 13  KB2  02-Nov-76  12:03 PM
Job number to attach to? 21
Password:
Attaching to Job 21
Failure to attach to Job 21
```

The user must try again. If the system successfully attaches the job to the terminal, the terminal becomes the console terminal of the job. Further terminal output is under programmed rather than system control.

To omit the printing of the identification and the query lines, simply type the job number on the same line as the ATTACH or ATT command as follows.

```
ATT 27
Password:
Attaching to Job 27

Ready
```

The READY message indicates that the attached job is at the system monitor level.

### 14.1.2 Running LOGIN at a Logged In Terminal

If the user types the HELLO or the ATTACH command at a terminal already logged into the system, the LOGIN system program is loaded into the user's job area and is started. The previous contents of the user's area are destroyed. LOGIN prints the system identification line with one additional item inserted. Between the job number and the keyboard number printed on the line, LOGIN inserts the project-programmer numbers of the account under which the current job is running. Typing the LOGIN, LOG, ATT or I command at a terminal already logged into the system causes the system monitor to print the ?WHAT? error message and the READY message, unless these have been installed as valid CCL commands.

LOGIN determines if any other jobs are running under the same account and prints the message informing the user of the number of those jobs. The following sample dialogue shows the procedure.

```
Ready

HELLO

RSTS V06B-02 Timesharing  Job 15  [1,210]  KB3  02-Nov-76  02:53 PM
1 other user is logged in under this account

Ready
```

If any such jobs are running detached, LOGIN also prints the message informing the user of the number of each such job and, on the following line, prints the ATTACH TO query as follows.

```
Ready

HELLO

RSTS V06B-02 Timesharing  Job 27  KB2  02-Nov-76  01:02 PM
Job 15 is detached under this account
Job number to attach to? 40
No job by that number - try again
Job number to attach to?
```

To attach one of the jobs to the terminal, the user types one of the job numbers reported in the message. LOGIN determines whether the job is nonexistent or whether it is already attached to another terminal. In either case, the program prints an appropriate error message saying try again and subsequently reprints the ATTACH TO query.

To continue running the current job, the user types the RETURN key in response to the ATTACH TO query. As a result, LOGIN prints the information message telling how many other jobs are running under the same account and prints the READY message. The system clears the LOGIN program out of memory.

```
Job number to attach to?
2 other users are logged in under this account


Ready
```

When the user responds to the ATTACH TO query by typing one of the job numbers reported in the message, the program proceeds as shown in the following sample printout.

```
HELLO

RSTS V06B-02 Timesharing  Job 36  [1,210]  KB3  02-Nov-76  02:55 PM
Job 15 is detached under this account
Job number to attach to? 15
Attaching to Job 15

Ready
```

The READY message indicates that the new job is at system command level.

To attach to a job known to be running detached under the same account, the user can type the job number on the same line as the ATTACH command. The LOGIN program determines if the job specified exists and is detached. If not, it prints an appropriate error message and the ATTACH TO query. The user can type another job number or the RETURN key. If the job exists and is detached, LOGIN compares the account numbers under which both the current job and the detached job are running. If the accounts are different, the program prompts the user for the password of the account under which the detached job is running. The following sample printout shows the procedure.

```
ATTACH 51
No Job by that number - try again
Job number to attach to? 15
Password:
Attaching to Job 15
Failure to attach to Job 15

Ready
```

After the user enters the password, the program prints the ATTACHING TO message and attempts to attach the detached job to the terminal. If the password is not valid, the program prints the FAILURE TO ATTACH and the READY messages and exits to the monitor, which clears the LOGIN program from the user's job area.

When the account numbers of the two jobs are the same, LOGIN omits the PASSWORD prompt message and attaches the job as shown below.

```
ATTACH 24
Attaching to Job 24

Ready
```

The READY message indicates that the job is at the system monitor level.

To change accounts without logging off the system, the user types the HELLO command followed by the account number. For example:

```
Ready

HELLO

RSTS V06B-02 Timesharing  Job 20  [2,227]  KB2  02-Nov-76  05:16 PM

Ready
```

```
HELLO 1/210
Password:

Ready

HELLO

RSTS V06B-02 Timesharing   Job 20   [1,210]   KB2   02-Nov-76   05:17 PM

Ready
```

To have the system print the system notice message, the user replaces the / character in the account number with a comma.

### 14.1.3 Running Other Programs from a Logged Out Terminal

Certain commands typed at a logged out terminal cause LOGIN to chain to another program in the system library. The system manager can modify the LOGIN program to recognize other commands and to chain to a program stored in the system library.

For example, it is convenient to determine job status without logging a job into the system. The following printout shows the procedure.

```
SYS/4

 4    [1,210] KB25     NONAME   16K    SL             0.1   BAS4F

Bye
```

LOGIN runs and recognizes the SYS command of the SYSTAT system program. LOGIN writes the option given in the command in the core common area and chains to the SYSTAT program at line 32000. SYSTAT reads the option from the core common area and prints the appropriate report. It then exits to the monitor, which prints the BYE message and clears the contents of memory.

## 14.2 LEAVING THE SYSTEM: THE LOGOUT PROGRAM

LOGOUT is called when the user has completed all processing and is ready to leave the terminal. The LOGOUT program is started when the BYE command is typed at a user terminal logged into the RSTS/E system. LOGOUT checks the current user's disk quota to ensure that the user does not log out of the system with more than the acceptable amount of disk storage being used for his files. If the user's disk files are within the acceptable disk quota size, LOGOUT disconnects the terminal from the system, removes the current job number from the list of active jobs and prints some information on the duration of the current job.

In response to the BYE command, LOGOUT prints:

```
CONFIRM:
```

The user can type any of the responses shown in Table 14-1.

**Table 14-1 LOGOUT CONFIRM: Responses**

| CONFIRM: Response | Meaning |
|---|---|
| Y | The system performs the checks described above. If successful, the LOGOUT messages are printed. If not successful, an error message is printed and the user must delete some files. |
| N<br>CTRL/C | These responses indicate that the user does not want to log out of the system. The LOGOUT procedure is terminated without logging the user off the system and the system prints the READY message. |
| ? | Causes LOGOUT to print an explanation of the acceptable responses to CONFIRM: |
| RETURN key | Causes LOGOUT to print a message instructing the user to type ? to obtain a description of logout procedures. |
| I | Causes LOGOUT to enter individual file deletion mode. |
| Other | Same as RETURN key. |
| F | Causes a fast logout procedure if user's disk storage space is within acceptable limits. |

In individual deletion mode, LOGOUT prints the name, size, protection code, and creation date of each file stored under the current user account number on the system disk. This information is followed by a ? after which the system awaits a response from the user which can be:

| File Deletion MODE Response | Meaning |
|---|---|
| RETURN key | Save the file just listed. |
| K | Delete (kill) the file just listed. |

An example of a LOGOUT sequence is shown below:

```
BYE Y
Disk quota of 400 exceeded by 52 blocks
Some file(s) must be deleted before logging out
Type '?' for help
Confirm: I
DATA   .001        200     60     03-Nov-76  ?
DATA   .002        150     60     03-Nov-76  ?
DATA   .003        100     60     03-Nov-76  ? K
Confirm: Y
Saved all disk files; 352 blocks in use, 48 free
Job 24 User 100,101 logged off KB3 at 03-Nov-76 03:09 PM
System RSTS V06B-02 Timesharing
Run time was 1.4 seconds
Elapsed time was 3 minutes, 59 seconds
Good afternoon
```

The user can omit the CONFIRM: message by typing the BYE command and the response to the CONFIRM: message. For example, to perform a fast logout, the user types

```
BYE F
```

The LOGOUT program runs and performs the fast logout procedure by printing a series of LINE FEED characters instead of printing the final accounting information. If the user job exceeds the acceptable limit for disk storage, LOGOUT prints the QUOTA EXCEEDED message and the CONFIRM: message to allow the user to delete some files before logging out.

# CHAPTER 15

# SYSTEM COMMUNICATION PROGRAMS

## 15.1 PRINTING A SYSTEM STATUS REPORT: THE SYSTAT PROGRAM

The SYSTAT program provides current system information in the areas of job, device, disk, and buffer status. SYSTAT can be called by a user logged into the system or from a terminal which is on-line but not logged into the system.

To start SYSTAT while logged into the system, the user types

```
RUN $SYSTAT
```

If the user is not logged into the system, he types

```
SYSTAT
```

which can be abbreviated to SYS.

If the user is already logged in, the system responds by printing:

```
OUTPUT STATUS TO?
```

at which point the user can indicate any RSTS/E device or a filename specification for the status report output. Possible replies by a user logged into the system are described below:

| SYSTAT Output Response | Meaning |
|---|---|
| LP: | send status report to the line printer if only one line printer is on the system or to line printer unit 0 if multiple line printers are on the system. |
| LPn: | send status report to line printer unit n if that printer is not currently in use. |
| KB: | send status report to the user terminal. (The RETURN key is equivalent to responding KB:) |
| KBn: | send status report to user terminal n in the system if that terminal is on-line and not currently in use. |
| PP: | send status report to the high-speed paper tape punch. |
| dev:filename.ext | send the status report to the file specified. The default device is the system device. No extension is appended unless specified by the user. |
| ? | send status report to a file on the public structure, and print the file's name. The file is named according to the current date and time of day; its extension is RPT. (The filename is explained in the following paragraph.) |

An example of an output file created by the ? response to the OUTPUT TO query is F04N59.RPT. The filename has 4 parts. The first is a letter from A to L, and denotes the month according to its alphabetical position; here, F, the sixth letter of the alphabet, denotes June. The second part is two digits from 01 to 31 denoting the day of the month; here, the fourth. The third part is a letter from A to X denoting the hour from 00 to 23; here, N denotes the 14th hour (2:00 p.m.). The fourth part is two numbers denoting the minute of the hour.

If SYSTAT is run by a user not logged into the system, the report is always sent to the user terminal requesting the report.

Following the device or filename specification, the user can specify one of the options in Table 15-1 to obtain a partial system status report. The option specifications are preceded by a slash if the user is logged into the system. The options can be typed following the SYS command if the user is not logged into the system.

**Table 15-1 SYSTAT Options**

| SYSTAT Option Specification | Meaning |
|---|---|
| /A | Report only status of attached jobs. |
| /B | Report only busy device status. |
| /D | Report only disk status. |
| /F | Report only free buffer status. |
| /Kn | Report only job status of terminal n in the system. |
| /M | Report only message receiver status. |
| /N | Report only status of non-privileged accounts. |
| /n | Report status of job n only. |
| /n,m | Report status of account [n,m] only. |
| /n,* | Report status of jobs with project number n only. |
| null | Report complete system status to include job, run-time system, busy device, disk structure, free buffer status, and message receiver statistics. |
| /P | Report only status of privileged accounts. |
| /R | Report only run-time system statistics. |
| /S | Report only job status. |
| /U | Report only status of unattached (i.e., detached) jobs. |
| /0,0 | Report only status of jobs not logged into the system. |

A minus sign ( - ) may be included with any option, in any position following the slash, to cause printing of an account number instead of [OPR] and [SELF].

The options S,A,B,D,F,N,P and U can be specified as separate options or in any combination. If multiple options are specified, only one slash is required.

The following examples are performed on a terminal logged into the system:

| | |
|---|---|
| RUN $SYSTAT<br>OUTPUT STATUS TO? STAT | creates complete system status report in the file STAT under the current account in the public structure. |
| RUN $SYSTAT<br>OUTPUT STATUS TO? LP: /3 | causes output of a status report for job 3 to the line printer. |
| RUN $SYSTAT<br>OUTPUT STATUS TO? /D | causes output of disk status report to the user terminal. |
| RUN $SYSTAT<br>OUTPUT STATUS TO? /SF | causes output of job and free buffer status to the user terminal. |

The following examples are performed on a terminal not logged into the system:

| | |
|---|---|
| SYS | causes output of complete system status report to the terminal. |
| SYS/D | causes output of the disk status report to the terminal. |
| SYS/BF | causes output of the busy device and free buffer status reports to the terminal. |
| SYS/5 | causes output of a status report for job 5 to the terminal. |
| SYS A/B | causes the ILLEGAL OPTION error since SYSTAT cannot create a file for a logged out job. |
| SYS /AP | causes output of a report of all attached, privileged jobs. |

### 15.1.1 Contents of the Status Report

A complete system status report is shown below:

```
RUN $SYSTAT
OUTPUT STATUS TO?
```

```
RSTS V06B-02 Timesharing status at 28-Oct-76, 03:47 PM Up: 22:50:46

Job    Who    Where    What    Size    State    Run-Time   RTS
 1    [OPR]   Det     ERRCPY    5K     SR         4:10.4  BAS4F
 2    [OPR]   Det     OPSRUN   16K     SL         8:28.8  BAS4F
 3    [OPR]   Det     QUMRUN   16K     SL        40:58.6  BAS4F
 4    [OPR]   Det     SPLIDL   16K     RN            0.8  BAS4F
 5    [OPR]   Det     SPLIDL   16K     SL D08        0.1  BAS4F
 6    [OPR]   KB44    NONAME    2K     ^C            9.4  BAS4F
 7    [OPR]   Det     BATIDL   13K     SL            0.1  BAS4F
 8    [OPR]   Det     BATIDL   13K     SL D16        0.1  BAS4F
 9    [OPR]   KB50    NONAME    2K     ^C            0.1  BAS4F
 11     1,208  KB36    PIP      14K     ^C A10     2:12.7  BAS4F
 13  232,17   KB49    NONAME    2K     ^C            5.0  BAS4F
 14     1,239  KB55    UTILTY   10K     ^C A06     3:53.9  BAS4F
 15  120,71   KB33    C3P351    3K     TT            6.2  BAS4F
 17  100,100  Det     VT5DPY   16K     SL        16:01.1  BAS4F
 18     1,240  Det     VT5DPY   16K     SL        11:23.7  BAS4F
 19   [OPR]   KB56    NONAME    2K     ^C A12    13:42.2  BAS4F
 20     1,201  KB25    TRNSCB    7K     RN         2:03.6  BAS4F
 21     1,204  KB52    BINCOM    4K     ^C            8.3  BASIC
 22   [OPR]   KB51    NONAME    2K     ^C A13       19.2  BAS4F
 23     1,227  KB24    NONAME    3K     ^C         2:55.8  BAS4F
 24   [SELF]  POJ20   SYSTAT    8K     RN Lck        1.7  BAS4F
 26  232,21   KB43    NONAME    2K     ^C         2:18.7  BAS4F
 27     1,202  Det     DEBUG     3K     SL           13.3  BAS4F
 28     1,201  Det     SPLIDL   16K     SL            0.1  BAS4F
 30   [OPR]   KB31    MODAS1    3K     ^C A09     2:24.4  BASIC

Busy Devices:
Device  Job     Why
PK0     20      INIT
LP0     4       AS+INIT
MM0     25      AS

Disk Structure:
Disk    Open    Free    Cluster Errors  Name    Comments
DS0        3     75         1      0     SWAP0   Pri
DS1        1                1      0             NFS
DK0        8     0          1      0     5002    Pub
DB1       17     14272      4      0     SYS881  Pub
DB2       13     14272      4      0     SYS882  Pub

Small   Large   Jobs    Hung TTY'S      Errors
270        1    25/63       0            481

Run-Time Systems:
 Name   Ext     Size    Users   Comments
BAS4F   BAC      15K     23     Perm, Addr:79, KBM, CSZ
BASIC   BAC      14K      2     Temp, Addr:94, KBM, CSZ
RT11    SAV       4K      0     Non-Res, KBM, CSZ, EMT:255
RTSLIB  TSK       4K      0     Non-Res
COBOL   TSK       4K      0     Non-Res

Message Receivers:
 Name   Job     Msgs    Max     Senders
ERRLOG   1       0       30      Priv
OPSER    2       0       30      Local
QUEMAN   3       0       60      Local
LP0SPL   4       0        5      Priv
LP1SPL   5       0        5      Priv
BA0SPL   7       0        5      Priv
BA1SPL   8       0        5      Priv
LP2SPL  28       0        5      Priv
```

The job status information includes a list of all active jobs by job number, the account number under which each job runs, the keyboard involved, the program name and size, the job state, the total amount of central processor run time exhausted, and the job priority. In the WHO column, SYSTAT substitutes OPR for the project-programmer number to denote an operator account. An operator job has a project number of 1 and a programmer number between 1 and 200. In the WHERE column, DET appears in place of the keyboard number for jobs which run detached from a keyboard. Also, the abbreviation PxJy appears for a job running on a pseudo keyboard. The value Px identifies pseudo keyboard unit x; and the value Jy denotes job number y, under which the controlling job is running. The SIZE column shows two numbers separated by a slash character. The first number is the current size in K words; the second number is the size to which the job can expand. The STATE column contains an abbreviation (see Table 15-3) telling the condition the job is currently in. The RUN-TIME column gives hours, minutes, seconds, and tenths of seconds of central processor time the job has consumed. The PRIORITY column (printed only if job is privileged), gives the number which the system assigns to determine the order in which to run jobs. Most jobs run at -8 priority; system jobs may run at special priorities of 0 or higher. The RTS column gives the name of the run time system under which the job is running.

The busy device status information reports devices which are assigned or opened by a specific user. Items reported are the device specification, the job owning that device, and the condition of the device (in the WHY column).

The disk status information describes each disk in use on the system. Items reported are: disk name (device specification), number of open files, number of free 512-byte blocks, pack cluster size, disk hardware error count, the pack identification or system logical name (if any) assigned for the device, and comments on its status. See Table 15-3 for abbreviations in the COMMENTS column.

The buffer status provides the following information: 1) the number of small (16-word) and large (256-word) buffers not currently running in use, 2) the number of jobs currently running and the maximum number allowed to run (two numbers separated by a slash), 3) the total number of errors logged on the system, and 4) a count of the number of times a hung terminal was found. A hung terminal is one that fails to respond to character transmission within a given time period.

The run-time system information gives the name of each run-time system, the default extension for (executable) files created by that run-time system, the size of the run-time system in K words, the number of user jobs currently executing under its control, and comments regarding its status. See Table 15-2 for abbreviations in the Comments column.

The message receiver report gives the receiving job's name and number, the number of messages queued for the job, and the declared maximum number of messages the job can have queued. It also tells whether local and network senders are allowed, and whether local senders must be privileged.

The abbreviations used in the SYSTAT report are defined in Table 15-2.

The message receiver report gives the receiving job's name and number, the number of messages queued for the job, and the declared maximum number of messages the job can have queued. It also tells whether local and network senders are allowed, and whether local senders must be privileged.

### 15.1.2 SYSTAT as a CCL Command

SYSTAT as a CCL command works similarly to SYSTAT typed at a logged out terminal. The CCL command, however, can contain a file specification. The following commands show the proper procedure.

```
SYS B

READY
```

**Table 15-2 SYSTAT Abbreviations**

| Abbreviation | Meaning |
|---|---|
| job status (states) | |
| DET | job is detached from all terminals |
| **,** | job is not logged into the system |
| OPR | job runs under a system operator account |
| SELF | job runs under current user account |
| RN | job is running or waiting to run |
| RS | job is waiting for residency |
| BF | job is waiting for buffers (no space is available for I/O buffers) |
| SL | job is sleeping (SLEEP statement) |
| SR | job is in a receiver sleep |
| FP | job is waiting for file processing action by the system |
| TT | job is waiting to perform output to a terminal |
| HB | job is detached and waiting to perform I/O to a terminal |
| KB | job is waiting for input from a terminal |
| ↑C | job is in CTRL/C state, awaiting input to the monitor |
| CR | job is waiting for card reader input |
| MT or MM | job is waiting for magtape I/O |
| LP | job is waiting to perform line printer output |
| DT | job is waiting for DECtape I/O |
| PP | job is waiting to perform output on the high-speed paper tape punch |
| PR | job is waiting for input from the high-speed paper tape reader |
| DK,DM,DB,DS, DP,DC,DF | job is waiting to perform disk I/O |
| DX | job is waiting for floppy disk I/O |
| RJ | job is waiting for RJ2780 I/O |
| ?? | job's state cannot be determined |
| The following status descriptions may appear after one or more of the other job state abbreviations: | |
| Lck | job is locked in memory for the current operation |
| Swi | job is currently being swapped into memory |
| Swo | job is currently being swapped out of memory |
| Xnn | job is swapped out and occupies slot nn in swapping file X; file is denoted by A,B,C, or D to represent files 0 through 3 of the swapping structure |
| busy device status | |
| AS | device is explicitly assigned to a job |
| INIT | device is open on a channel |
| DOS | magtape is assigned with DOS labeling format |
| ANSI | magtape is assigned with ANSI standard labeling format |
| disk status | |
| PUB | cartridge or pack is public |
| PRI | cartridge or pack is private |

(continued on next page)

Table 15-2 (Cont.) SYSTAT Abbreviations

| Abbreviation | Meaning |
|---|---|
| NFS | disk is open as non-file structured device |
| R-O | disk unit is read-only (write-locked) |
| DLW | date of last write (modify), rather than date of last access, is stored in file accounting entries |
| Lck | disk is in a locked state |
| run-time system status | |
| Non-Res | run-time system is non-resident |
| Loading | run-time system is being loaded into memory |
| Temp | run-time system will be removed from memory when not being used |
| Perm | run-time system will stay in memory when not being used |
| Addr:xxx | denotes the run-time system's starting address |
| KBM | run-time system can serve as keyboard monitor |
| 1US | run-time system can serve only 1 user |
| R/W | run-time system allows read/write access |
| NER | errors occurring within run-time system will not be sent to system error log |
| Rem | run-time system will be removed from memory as soon as all its jobs switch to another run-time system |
| CSZ | proper job image size (in K words) to run a program can be computed as K-size=(filesize+3)/4 |
| EMT:yyy | denotes the EMT code for special EMT prefix |
| message receiver status | |
| Local | local senders are allowed for this receiver ID |
| Priv | local senders must be privileged to send to this receiver ID |
| Network | network senders are allowed for this receiver ID |

SYSTAT creates file B and writes to it a complete system status report. The file resides under the current user account in the public structure.

```
SYS /B
```

SYSTAT prints a busy device status report at the terminal. To create a status report in a file, the user types the file specification with an option as follows.

```
SYS B/B

READY
```

SYSTAT creates a busy device status report in file B.

## 15.2 OBTAINING A DISK QUOTA REPORT: THE QUOLST PROGRAM

The QUOLST system program allows the user to determine what portion of his disk quota is currently occupied and the number of free blocks remaining on the system disk. QUOLST is called as follows:

```
RUN $QUOLST
```

Output from QUOLST includes the user account number and information printed under the following headings:

**Table 15-3 QUOLST Column Headings**

| Column Heading | Meaning |
|---|---|
| STR | STRucture, device being reported. |
| USED | number of used 256-word blocks under the user account. |
| FREE | number of free blocks remaining in the user account disk quota. |
| SYSTEM | number of free blocks remaining to the system on the structure indicated. |

Output from QUOLST looks like this:

```
RUN $QUOLST
QUOLST   V06B-03 RSTS V06B-02 Timesharing

USER:    [200,57]
STR      USED              FREE              SYSTEM
SY:      60                4940              5452

Ready
```

In this example, the user is logged into the system under account [200,57] and has used 60 blocks on the public disk structure with a quota of 5000 blocks (5000 – 60=4940 free blocks). There are 5452 free blocks on the public disk structure.

## 15.3 OBTAINING ACCOUNT DATA: THE MONEY PROGRAM

MONEY is the RSTS/E system accounting program which allows a user to obtain printed data on his own account status. The program is called as follows (only by a user logged into the system):

```
RUN $MONEY
```

In the following example, the user is logged into the system under account [100,100], and runs the MONEY program:

```
RUN $MONEY
MONEY   V06B-03 RSTS V06B-02 Timesharing
SYSTEM ACCOUNTING PROGRAM

 ACCT    PASSWORD     CPU-TIME    KCT'S CONNECT DEVICE   DISK  QUOTA  UFD
100,100                   26.8     2436       23      2    316   5000   16

Ready
```

The headings and information contained in the MONEY report are described in Table 15-4.

**Table 15-4 The MONEY Report**

| Heading | Information |
|---|---|
| ACCT | The user account number |
| PASSWORD | Not printed for non-privileged users |
| CPU-TIME | The total number of seconds of central processor time used by the account |
| KCT's | The total number of "kilo-core ticks" used by the account. This is a measure of total central processor usage, along with central processor time and memory usage. Whenever a program uses one-tenth of a second of CPU time, this value is incremented by the size of the program in K words. |
| CONNECT | The total number of minutes the account is logged into the system. |
| DEVICE | The total number of minutes devices are assigned by this account. |
| DISK | The number of blocks used on the public structure. |
| QUOTA | The disk quota in blocks. |
| UFD | The cluster size of the user file directory. |
| These values do not reflect the current timesharing session, because the accounting information is updated when the job is logged off. | |

## 15.4 SENDING A MESSAGE TO THE SYSTEM MANAGER: THE GRIPE PROGRAM

The GRIPE system program allows a user to communicate comments to the system manager. Comments which the user types while running GRIPE are written to a common file where they are retained for inspection by the system manager.

The user runs GRIPE by typing the following command:

```
    RUN $GRIPE
```

GRIPE indicates that it is ready to accept user comments by printing a query line as follows:

```
    YES?    (END WITH ESCAPE)
```

The user is then allowed to type the text of his comment which is entered into the common file. The user terminates the text of his comment by typing the ESCAPE or ALTMODE key. (Typing the ESCAPE or ALTMODE key is echoed at the terminal by a dollar sign ($) being printed. No carriage return-line feed operation is performed. The program indicates its acceptance of the text and its termination by printing the following lines.

```
    THANK YOU

    READY
```

## 15.5 DECLARING A TERMINAL IN USE: THE INUSE PROGRAM

The INUSE system program prints or displays the words IN USE in block letters on the terminal to warn others not to use it. This message is followed by the user's job number and account number.

INUSE is called as follows:

    RUN $INUSE

The printout from this program is shown below.

```
RUN $INUSE
  IIIIII    NN      NN              UU      UU    SSSSSS    EEEEEEEEEE
  IIIIII    NN      NN              UU      UU    SSSSSS    EEEEEEEEEE
    II      NNNN    NN              UU      UU  SS      SS  EE
    II      NNNN    NN              UU      UU  SS      SS  EE
    II      NNNN    NN              UU      UU  SS          EE
    II      NNNN    NN              UU      UU  SS          EE
    II      NN  NN  NN              UU      UU    SSSSSS    EEEEEE
    II      NN  NN  NN              UU      UU    SSSSSS    EEEEEE
    II      NN    NNNN              UU      UU          SS  EE
    II      NN    NNNN              UU      UU          SS  EE
    II      NN    NNNN              UU      UU  SS      SS  EE
    II      NN    NNNN              UU      UU  SS      SS  EE
  IIIIII    NN      NN                UUUUUU      SSSSSS    EEEEEEEEEE
  IIIIII    NN      NN                UUUUUU      SSSSSS    EEEEEEEEEE

        BY JOB 24 USER [2,201]
```

To regain control of the terminal, the user types the CTRL/Z or CTRL/C combination, or any valid command.

# CHAPTER 16

# FILE UTILITY PROGRAMS: LISTING, EDITING, AND READING FILES

## 16.1 LISTING DIRECTORY OF FILES: THE DIRECT PROGRAM

The DIRECT program lists file related information from a disk directory. The benefits of DIRECT are increased speed and more options compared with other methods of listing directories. DIRECT opens a user's directory as a file and reads information by immediately accessing the blocks in the directory. This action is faster than the conventional method of passing the request for such information to internal system functions. Since the program follows pointers through a disk directory, incorrect information may be printed if the pointers are changed during program execution. For example, if a file is opened on the current account by another user, then the information printed may be incorrect.

The user runs DIRECT by typing the RUN $DIRECT command or by using the CCL command explained later in this section. When DIRECT runs as a result of the RUN command, it prints a header line and the # character, which acts as a prompt. The user can then type a command to DIRECT. If the user runs DIRECT by the CCL command, he includes the command to DIRECT in the CCL command.

The general format of the command is as follows:

```
output=input/option(s), input/option(s), . . .
```

Output is optional and can be a device specification or a disk file specification. If output is not given, the = character is optional and DIRECT prints output at the user's terminal. If an extension does not appear with an output filename, DIRECT appends .DIR unless the user forces a null extension by specifying the . character with the filename. Input can be any number of full disk file specifications. The full disk file specification on input can include a device, filename, extension and project-programmer number. If a device is not given, DIRECT uses the public structure and denotes it by the SY: specification.

The filename, extension, and project-programmer fields can contain * and ? characters to denote wild card specifications. If no file specification is given or if an * character is given as the file specification, DIRECT processes all files in the directory. DIRECT applies the default interpretations shown for the following specifications for a given directory.

| **User Types:** | **Program Interprets as:** | **Meaning:** |
|---|---|---|
| null | *.* | All files |
| * | *.* | All files |
| *. | *. | All files with null extensions |
| . | *. | All files with null extensions |
| .EXT | *.EXT | All files with extension EXT |
| FILE | FILE.* | All files with filename FILE |

With a file specification, the user can specify one or more options. If no options appear, DIRECT proceeds as if the user had specified /DI. Table 16-1 lists and describes the options.

**Table 16-1 DIRECT Options**

| Type | Format | Meaning |
|---|---|---|
| Individual | /NA | List filenames only. |
| | /EX | List filenames and extensions of each file. |
| | /SI | List filename, extension and list size of each file as number of 512-byte blocks occupied. |
| | /PR | List filename, extension and file protection code. |
| | /LA | List filename, extension and date of last access for each file. |
| | /DA | List filename, extension and date of creation for the file. |
| | /TI | List filename, extension, date and time of day when file was created. |
| | /CL | List filename, extension, and file cluster size. |
| | /SU | List only summary data to include number of designated files and total number of blocks occupied by designated files. |
| Aggregate | /BR<br>/F | List filenames and extensions with a brief summary message. |
| | /DI:S<br>/S | List all relevant data to include headings, filenames, extensions, size, protection code, date of last access, date of creation, time of creation, clustersize, associated run-time system, file attributes if any, and summary. (Called slow directory.) |
| | /DI<br>null | List most important data to include heading, filename, extension, size protection code, date of last access and summary. |
| General | /HD | Print heading at top of columns on the listing. |
| | /W | List data across the width of a line rather than one item per line. Useful with large directory listings and individual options. |
| | /HE | Print the file DIRECT.HLP which describes the DIRECT program. |
| | /BK | List the directory for the specified device in reverse chronological order. As a result, the most recently created files appear at the beginning of the listing. If /BK is used to list files in the public structure and multiple disks are in the public structure, the listing reflects reverse order for each disk. |

To list a directory at the line printer, the user specifies the device designator and options in the command. For example,

```
#LP0:=*.BA?/F/W
#
```

The command lists filenames and extensions of all files with BA as the first 2 characters of the extension. DIRECT formats the data across the width of the line printer paper.

To list directories of several accounts and place them in a disk file, the user specifies the project-programmer field and the disk file specification in the command. For example,

```
#PROJ=[120,*]/DI
#
```

DIRECT creates the file PROJ.DIR in the current account and writes, to the file, directory listings of all accounts with project number 120.

An error encountered in a command causes DIRECT to print a message followed by the # character. The user must retype the entire command correctly. The messages are described in Table 16-2.

**16.1.1 DIRECTORY as a CCL Command**

The following commands show some useful methods of requesting listings with the standard CCL command DIRECTORY or its abbreviation DIR. To list the filenames and extensions of all files in the current account, the user types the following command.

```
DIR /BR/W
```

DIRECT prints the listing at the user's terminal and lists the information across the width of the page. To list at another device the filenames, extensions, sizes, protection codes and creation dates for certain files in the current account, the user types the following command.

```
DIR LP1:=*.BAS
READY
```

DIRECT prints the listing of all BASIC-PLUS source files on line printer unit 1. The READY message indicates DIRECT has completed printing. To obtain a reverse listing, the user types the following command.

```
DIR /DI/BK
```

DIRECT prints, at the current terminal, directory and summary information in reverse chronological order. To print the file DIRECT.HLP on the line printer, the user types the following command.

```
DIR LP1:=/HE
READY
```

**Table 16-2 DIRECT Program Error Messages**

| | |
|---|---|
| ?DEVICE NOT DIRECTORY STRUCTURED | Device specified does not use a directory for file access. |
| ?DIRECTORY OF dev:[n,m] IS EMPTY | DIRECT finds the account [n,m] on device dev: contains no entries. |
| ?DISK PACK IS NOT ON-LINE | The pack or cartridge referred to is either not mounted or is off-line. |
| ?ILLEGAL FILE NAME <filename> | The file specified by <filename> contains a logical device name which the user has not reserved by the ASSIGN command. |
| ?ILLEGAL INPUT FILE SPEC <file spec> | The file specification indicated by file spec generates the error ?ILLEGAL FILE NAME. |
| ?ILLEGAL SWITCH text | File specification contains an undefined option indicated by text. |
| ?INVALID DEVICE SPECIFICATION | The device specification is invalid or the device referred to does not exist on the system. |
| ?NO DIRECTORY FOR [n,m] ON dev: | DIRECT cannot find an account for user account [n,m] on the device dev: or else DIRECT encounters a protection violation. |
| ?NO HELP AVAILABLE | The file DIRECT.HLP is not on the system library account. |
| ?NO SUCH FILE AS <file spec> ON [n,m] | DIRECT cannot find the requested file indicated by <file spec> in the account [n,m]. |
| ?OUTPUT FILE MUST BE IN THE USER'S AREA | DIRECT does not create an output disk file in another account if user is not privileged. |
| ?TOO MANY FILES FOR INVERTED DIRECTORY LISTING | DIRECT limits use of the /BK option to accounts with less than 200 files. |

## 16.2 EDITING FILES: THE EDIT PROGRAM

The EDIT system program is used to prepare and modify text or program files. It can be run by any user. To run EDIT, the user types:

```
RUN $EDIT
EDIT   V06B-03   RSTS V06B-02   TIMESHARING

#
```

In response to the number sign (#), the user must specify the input files he wishes to modify and the files to be created as output. The form of this specification is:

```
#OUT1,OUT2=IN1,IN2/B
```

where IN1 and OUT1 are the primary input and output files, respectively, and IN2 and OUT2 are the secondary input and output files. These file names may be any valid RSTS/E file specifications, including a device, name, extension, project-programmer number, and protection code. The /B is included if the user desires to edit a BASIC-PLUS program file containing LINE FEED continuation of lines. (See the *BASIC-PLUS Language Manual*, Section 2.3.2.) Only one of these file specifications – the primary output file OUT1 – must be specified; if specified alone, it indicates the creation of a new file.

If IN1 and OUT1, the primary input and output, are specified as the same file, the primary input file will be re-named after the editing is complete to have an extension of .BAK, designating it as the "backup" file. In this case EDIT will use a temporary name of EDITnn.TMP for the primary output file during editing operations (where "nn" is the job number under which EDIT is being executed). If the secondary output file is a line printer (LPn:), the printer will be assigned only when needed for output. (If the printer is unavailable when needed, the user is given the option of waiting for some time of his choosing or of aborting the output request.)

Once EDIT has been given the file specifications, it will open the necessary files and respond with an asterisk (*), indicating it is ready to accept editing commands from the user. The valid commands are as described in the *RSTS/E Text Editor Manual,* and are summarized in Table 16-3.

The EDIT-11 command T (trailer) has no meaning to the RSTS/E EDIT program. All other descriptions of EDIT-11 commands and how they operate are valid for RSTS/E EDIT as well. If an error occurs in executing a command as typed by the user, the command already executed will be printed, terminated with a question mark (?) at the point at which the error was noted. When editing is complete, EDIT returns to the request for file specifications (#).

### 16.2.1 EDIT as a CCL Command

EDIT may be run as a CCL command by typing one of the following:

1. EDIT
2. EDIT OUT1,OUT2=IN1,IN2
3. EDIT FILENAME
4. CREATE FILENAME

(EDIT may be minimally abbreviated to ED, and CREATE to CRE.)

The first of these is equivalent to RUN $EDIT. The header line is printed, and a # character prompts the user to specify his files. The second CCL command, shown above, runs EDIT and also automatically sets up the input and output files as specified. No header line is printed.

The third CCL command is equivalent to the set of commands:

```
RUN $EDIT
FILENAME =FILENAME/B
R
```

except that the header line of EDIT is not printed. EDIT runs, sets up a .BAK file, reads the first buffer, and waits for the user to type editing commands.

The last CCL command shown above sets up the file named FILENAME as an output file. No .BAK file is set up; there is no input file; and no header is printed. If the file existed previously, it is reduced to zero length. As a first command, the user types I (insert).

The CREATE command is equivalent to the instructions:

```
RUN $EDIT
FILENAME
```

**Table 16-3 Summary of EDIT Program Commands**

| Command | Format[1] | Result |
|---|---|---|
| Read | R | Read from primary input file until form feed is encountered or all internal buffer space is filled. |
| Edit Read | ER | Read from secondary input file until form feed is encountered or all internal buffer space is filled. |
| Write | nW | Write n lines into primary output file, starting from the current position of Dot. |
| Edit Write | nEW | Write n lines into secondary output file, starting from the current position of Dot. |
| Next | nN | Write the contents of the Page Buffer into the primary output file, kill the buffer, and read a page of text from the primary input file. Repeat n times. Equivalent to B/W/DR. |
| Form feed | nF | Insert n form feed characters at Dot. |
| Beginning | B | Move Dot to the beginning of the Page Buffer. |
| Advance | nA | Advance Dot n lines. Leaves Dot at beginning of line. |
| Jump | nJ | Move Dot over n characters. |
| Delete | nD | Delete n characters from text, starting at Dot. |
| Kill | nK | Kill n lines of text, starting at Dot. |
| Mark | M | Mark the current location of Dot. |
| Save | nS | Save the next n lines in the Save Buffer, starting at Dot. |

[1]In the table, # represents any ASCII character. <CR> represents a return character, and <LF> represents a line feed character.

(continued on next page)

**Table 16-3 (Cont.) Summary of EDIT Program Commands**

| Command | Format[1] | Result |
|---|---|---|
| Unsave | U | Copy the contents of the Save Buffer into Page Buffer at Dot. |
| List | nL | List n lines on teleprinter, starting at Dot. |
| Verify | V | Verify the present line via teleprinter. |
| Get | nG#XXXXX# or nG<CR> XXXX<CR> <LF> | Search the current buffer for the nth occurrence of XXXXX. Return with Dot following XXXXX. |
| wHole | nH#XXXXX# or nH<CR> XXXX<CR> <LF> | Search the remainder of the input file for nth occurrence of XXXXX. If found on this page, return with Dot following XXXXX. If not found, execute an N command and continue search. |
| Edit wHole | nEH#XXXXX# or nEH<CR> XXXX<CR> <LF> | Perform a wHole search for the nth occurrence of XXXXX, using the secondary input and primary output files. |
| Position | nP#XXXXX# or nP<CR> XXXX<CR> <LF> | Perform a Next command, then search for the nth occurrence of XXXXX. If found, return with Dot following XXXXX. If not found, clear the buffer, read another page, and continue search. |
| Edit Position | nEP#XXXXX# or nEP<CR> XXXX<CR> <LF> | Perform a Position search using secondary input rather than primary input file. |
| Insert | I#XXXXX# or I<CR> XXXX<CR> <LF> | Insert the text XXXXX at Dot. Move Dot to follow XXXXX. |
| Change | nC#XXXXX# or nC<CR> XXXX<CR> <LF> | Change n characters starting at Dot to XXXXX. Equivalent to Insert followed by n Delete. |

[1] In the table, # represents any ASCII character. <CR> represents a return character, and <LF> represents a line feed character.

(continued on next page)

**Table 16-3 (Cont.) Summary of EDIT Program Commands**

| Command | Format[1] | Result |
|---|---|---|
| eXchange | nX#XXXXX# or nX<CR> XXXX<CR> <LF> | eXchange n lines starting at Dot for XXXXX. Equivalent to Insert followed by n Kill. |
| Execute Macro | nEM | Execute the first line of the Save Buffer as a command string n times. |
| Exit | EX | Perform consecutive Next commands until end of primary input file is reached.<br><br>Close all files, and return to file specification request. |
| Edit Open | EO | Move to beginning of the secondary input file. |
| End File | EF | Close the primary output file to any further output and close the primary input file. |

[1]In the table, # represents any ASCII character. <CR> represents a return character, and <LF> represents a line feed character.

## 16.3 COMPARING FILES: THE FILCOM PROGRAM

The FILCOM (file compare) system program compares two ASCII files, line by line, and prints the differences found. The user must specify which two files are to be compared as well as how many successive lines must be compared at a time. This second consideration is especially important for isolating differences in program subroutines.

FILCOM is called as follows:

```
RUN $FILCOM
```

The first query line printed is:

```
OUTPUT TO?
```

The user types the device designator of the peripheral device on which the comparison information is to be printed. Typing the CR key alone specifies the keyboard on which the user is working.

If the user includes /P with a file name in response to the OUTPUT TO query, FILCOM creates the output file as a patch file. With the APPEND command, such a patch file can later be included in the file specified as INPUT FILE#1. This action makes the resultant file equivalent to the BASIC-PLUS program specified as INPUT FILE#2. To create the patch file, the user must also answer both the BASIC+ LINES and BLANK LINES query with YES.

The next two printed query lines request the names of the files to be compared. The user should be sure to use the full name of each file, including its extension. For example:

```
INPUT FILE#1? SORT1.BAS

INPUT FILE#2? SORT2.BAS
```

Any number of lines can be compared at a time. The number of successive lines that determines a match is specified by the user response to the following FILCOM query:

```
HOW MANY TO MATCH?
```

Responses to this query are discussed in detail later in this section. If the CR key is typed alone, the number of successive lines to match is automatically assumed to be three.

The next query line is:

```
BASIC+ LINES?
```

The user types the letter Y in response to this query to instruct the program to consider BASIC-PLUS continuation lines as part of the numbered line. In this way, multiple line statements can be compared. For example, consider the line shown below.

```
100 FOR J=10. TO 15.
     \ PRINT 3.14*J/2.
     \ NEXT J
```

If the user types Y in response to the BASIC+ LINES query, FILCOM will compare all three statements (FOR, PRINT and NEXT) as a single line.

If the user responds with any other character (or no character), FILCOM will compare only the statement printed on the first line (i.e., FOR J=10. TO 15) to a single line in the other file. Typing Y ensures that FILCOM prints the line numbers of any BASIC-PLUS lines found to be different.

The next query line is

BLANK LINES?

Typing the letter Y in response to this question instructs the program to compare blank lines. If the Y is not specified, FILCOM skips all blank lines.

Assume the following two files are on the user's disk:

```
TEST1
10  REM A
20  REM B
30  REM C
40  REM D
50  REM E
60  REM F
70  REM G
80  REM H
90  REM I
100  REM J
110  REM K
120  REM L
130  REM M
140  REM N
150  REM O
160  REM P
170  REM Q
180  REM R
190  REM S
200  REM T
210  REM U
220  REM V
230  REM W
240  REM X
250  REM Y
260  REM Z
```

```
TEST2
10  REM A
20  REM B
30  REM C
70  REM G
80  REM H
90  REM I
100  REM J
110  REM 1
120  REM 2
130  REM 3
140  REM N
150  REM O
160  REM P
170  REM Q
180  REM R
190  REM S
200  REM T
210  REM U
220  REM V
222  REM 4
224  REM 5
230  REM W
240  REM X
250  REM Y
260  REM Z
```

EXAMPLE 1

To compare these two files, TEST1 and TEST2, FILCOM is run as follows:

```
RUN $FILCOM
FILCOM  V06B-03 RSTS V06B-02 Timesharing
FILE COMPARISON PROGRAM
Output to?
Input File #1? TEST1.BAS
Input File #2? TEST2.BAS
How Many to Match?
BASIC+ Lines?
Blank Lines?
********************
1) TEST1.BAS
40 REM D
```

```
50 REM E
60 REM F
70 REM G
*****
2) TEST2.BAS
70 REM G
********************
1) TEST1.BAS
110 REM K
120 REM L
130 REM M
140 REM N
*****
2) TEST2.BAS
110 REM 1
120 REM 2
130 REM 3
140 REM N
********************
1) TEST1.BAS
230 REM W
*****
2) TEST2.BAS
222 REM 4
224 REM 5
230 REM W

?3 Differences Found

Ready
```

Notice that FILCOM prints the lines, from both files, that do not compare, followed by the first line that does compare. The first line in each group is the first line that does not compare and the last line in each group is the first line that does compare. In general, the program does not print groups of lines that are identical. FILCOM first prints line 40 of TEST1, since it is the first line that does not appear in TEST2. This line is considered the first difference. FILCOM continues to print TEST1 lines until it locates 3 lines that can be matched against 3 lines in TEST2. In this case, lines 70, 80, and 90 of both TEST1 and TEST2 are identical, so the printout for this difference is ended.

Since lines 80 through 100 are identical in both files, FILCOM does not print them. The next lines that do not compare are lines 110 through 130. These lines are printed under both files. This is considered the second difference. FILCOM continues to print lines until it locates 3 matching lines. Lines 140 through 160 are identical for both files.

Since lines 150 through 220 are identical in both files, FILCOM does not print them. The next lines that do not compare are 222 and 224 of TEST2. Consequently, they are printed under the TEST2 heading. This is considered the third difference. FILCOM continues to print TEST2's lines until it locates 3 lines that can be matched against 3 lines in TEST1. Lines 230 through 250 are identical for both files.

Finally, FILCOM prints the number of differences found. If one of the two files compared was empty (e.g., an empty data file), FILCOM would have printed A NULL FILE??

EXAMPLE 2

Files TEST1 and TEST2 can be compared in another way. To compare these files for eight consecutive identical lines instead of three, FILCOM is run as follows:

```
RUN $FILCOM
FILCOM  V06B-03 RSTS V06B-02 Timesharing
FILE COMPARISON PROGRAM
Output to?
Input File #1? TEST1.BAS
Input File #2? TEST2.BAS
How Many to Match? 8
BASIC+ Lines?
Blank Lines?
********************
1) TEST1.BAS
40 REM D
50 REM E
60 REM F
70 REM G
80 REM H
90 REM I
100 REM J
110 REM K
120 REM L
130 REM M
140 REM N
*****
2) TEST2.BAS
70 REM G
80 REM H
90 REM I
100 REM J
110 REM 1
120 REM 2
130 REM 3
140 REM N
********************
1) TEST1.BAS
230 REM W
*****
2) TEST2.BAS
222 REM 4
224 REM 5
230 REM W

?2 Differences Found


Ready
```

In this case, the first difference found, again, is line 40 in TEST1. Instead of matching lines 70 of both files, FILCOM begins to match line 140, since 140 to 220 is the next group of at least eight successive identical lines. Although lines 70 through 100 are identical in both files, they form less than eight successive lines.

Now assume the following two files are on the user's disk:

| FILE 1 | FILE 2 |
|---|---|
| THIS IS LINE 1 | THIS IS LINE 1 |
| THIS IS LINE 2 | THIS IS LINE 2 |
| | |
| THIS IS THE LINE AFTER THE BLANK LINE | |
| THIS IS THE NEXT LINE | THIS IS THE LINE AFTER THE BLANK LINE |
| | THIS IS THE NEXT LINE |

Notice that file 2 has an extra blank line.

EXAMPLE 3

To compare these two files, FILCOM is run as follows:

```
RUN $FILCOM
FILCOM  V06B-03 RSTS V06B-02 Timesharing
FILE COMPARISON PROGRAM
Output to?
Input File #1? FILE1
Input File #2? FILE2
How Many to Match?
BASIC+ Lines? NO
Blank Lines? NO

0 Differences Found
```

Since NO was typed after the query BLANK LINES?, FILCOM did not compare blank lines. Typing the letter Y in response to the BLANK LINES query, however, results in the printout shown below.

```
RUN $FILCOM
FILCOM  V06B-03 RSTS V06B-02 Timesharing
FILE COMPARISON PROGRAM
Output to?
Input File #1? FILE1
Input File #2? FILE2
How Many to Match?
BASIC+ Lines? NO
Blank Lines? YES
********************
1) FILE1
THIS IS THE LINE AFTER THE BLANK LINE
*****
2) FILE2

THIS IS THE LINE AFTER THE BLANK LINE

?1 Difference Found


Ready
```

**NOTE**

If too many differences are found, the program will abort and print the error message ?MAXIMUM MEMORY EXCEEDED.

# CHAPTER 17
# DEVICE UTILITY PROGRAMS: TRANSFERRING FILES BETWEEN DEVICES

## 17.1 DEVICE TRANSFER: THE PIP PROGRAM

The PIP (Peripheral Interchange Program) system program performs disk and peripheral device transfers as well as several other file utility functions. Two forms of the PIP program are available; their only difference is that one runs in less than 8K words of memory and is generally less powerful than the larger version, which requires 16K words to run. Refer to Section 17-2 for a description of the larger version of PIP. (RSTS/E PIP commands, wherever possible, have been made compatible with DOS/BATCH-11 PIP commands.)

PIP can be called by users logged into the system as follows:

```
RUN  $PIP
```

PIP responds by identifying itself and printing a pound sign (#) to indicate that it is able to accept input commands:

```
PIP    V06B-03    RSTS V06B-02    TIMESHARING
#
```

In order to return to BASIC-PLUS command level, the user types CTRL/C or CTRL/Z. The system responds by printing READY. For example:

```
#↑Z

READY
```

A CTRL/Z is equivalent to an end-of-file on the user terminal and causes an orderly exit from PIP when the current operation is complete. Typing CTRL/C causes an immediate exit from PIP; it does not complete the operation in progress, although it leaves the directories in an orderly state.

### 17.1.1 PIP Command Line Specifications

Spaces and tabs within a PIP command line are ignored. PIP commands must be typed on a single line and be no more than 80 characters long.

Output file specifications are of the form:

```
dev: [proj,prog] name.ext<prot>
```

The elements of the output file specification are described in Table 17-1. Input file specifications are of the form:

```
dev: [proj,prog] name.ext
```

Elements of the input file specification are described in Table 17-2.

With PIP there is at most one output file, but there may be any number of input files. Where more than one input file is specified, all filenames which are not preceded by a device specification are assumed to be on the system disk. A null file specification duplicates the immediately preceding file specification in all details.

PIP options are specified in the form:

```
/option:argument
```

Table 17-1 Output File Specification Elements

| Element | Description | Default |
|---|---|---|
| dev: | device specification | SY: |
| [proj,prog] number | account specification, project-programmer number | current user account |
| name | filename specification[1] | none |
| .ext | filename extension[1] | none |
| <prot> | protection code | <60>, equivalent to read and write protect against everyone but the owner |
| no specification | | KB:PIP.OUT |

[1] Output-only devices (non-file-structured devices) such as KB:, LP: and PP: ignore the filename and extension specifications.

Table 17-2 Input File Specification Elements

| Element | Description | Default |
|---|---|---|
| dev: | device specification | system disk (DF:,DK:,DP:, or DB:) |
| [proj,prog] | account specification, project-programmer number | current user account number |
| name | filename specification[1] | none |
| .ext | filename extension[1] | .BAS (BASIC-PLUS source program) |
| no specification | | last file specification is used again. Initial default is SY: |

[1] Input-only devices (non-file-structured devices) such as KB:, PR:, and CR: ignore the filename and extension specifications.

Options are always begun with a slash and terminated with a comma (,), left angle bracket (<), another slash (/), or a line terminating character (RETURN or ESCAPE). The option argument is a function of the individual operation to be performed. The default option is a formatted ASCII file transfer.

## 17.1.2 File Transfers Including Merge Operations

File transfer and/or file merge operations take the following command format:

```
#output file=inputfile(s)/option
```

Only one output file can be specified. If one input file is specified, a copy of that file is transferred to the output file specification (the original input file remains unchanged). If more than one input file is specified, copies of the file are merged into a single output file (the original input files remaining unchanged). The options available on a file transfer and/or merge operation are described in Table 17-3.

**Table 17-3 File Transfer and Merge Options**

| Option | Function |
|---|---|
| no option | ASCII file transfer is performed. |
| /FA | Formatted ASCII transfer is performed (Nulls, parity bits,[1] and RUBOUT's are ignored). |
| /BL | Block mode transfer is performed using the default block sizes. |
| /BL:n | Block mode transfer is performed using a block which is n bytes long. |
| /CO | Contiguous mode transfer is performed with null fill characters inserted into any partial buffer remaining. Must be used when transferring a .BAC file, a virtual core array, or a file created by Record I/O, from disk to DECtape. |
| /CO:T | Contiguous mode transfer is performed with any partial buffer remaining being truncated. Must be used when transferring a .BAC file, a virtual core array, or a file created by RECORD I/O from DECtape to disk. |
| /CL:n | Set output cluster size to n. |
| /GO | Ignore ?USER DATA ERROR ON DEVICE errors. |
| /HE | Appends "$PIP.TXT" to command line to have PIP output the helping text explaining PIP commands and options. |
| /UP | Update file transfer (equivalent to OPEN AS FILE used for output files). |
| /RW:NO | Disable rewinding of magtape before and after a file transfer. |

[1] Parity bits are associated with some ASCII codes, making each ASCII character 8 bits long rather than 7 bits. Even parity implies that the number of bits set within the 8 bit field is an even number.

The following is an example of multiple file transfer:

```
#DT1:FILET.BAS=FILE1,FILE2,PR:,DT0:FILE3
```

This command transfers, to the file FILET.BAS on DECtape unit 1, the following files: FILE1.BAS, FILE2.BAS, 1 file from the high speed paper tape reader, and FILE3.BAS from DECtape unit 0.

Since a PIP command must be typed on a single line, the number of files which can be merged into a single file is limited only by a command string length of 80 characters.

### 17.1.3 Changing Filename or Protection Code

PIP can be used to change a filename specification without transferring any data. The general format for such a command is as follows:

```
#new file specification=old file specification/RE
```

To change the filename, extension, and/or protection code of a stored file, the user types the new file specification (including the device specification and, optionally, the protection code), an equal sign (=), the current file specification, and the option /RE. The user need not indicate the current protection code. In the following example, the user changes the filename, extension, and protection code.

```
#FILE.EXT<40>=ABC.DAT/RE
```

In the following example, only the filename is changed. The protection code for the file MAT.BAC remains the same as it was for ORIG.BAC.

```
#DM1:MAT.BAC=DT1:ORIG.BAC/RE
```

In the case where only the protection code is to be changed, a new filename specification need not be typed. A shorter form is:

```
#=old file specification<prot>/RE
```

This command string format indicates that the file protection is to be updated. For example:

```
#=MAG.BAS<48>/RE
```

Notice that the device specification on both sides of the equal sign must be the same. If the aim is to move the file from one device to another, the file copy technique (see Section 17.1.2) is used. Note also that PIP assumes a default filename extension of .BAS on input files, but an extension must be specified on output files or none is appended.

### 17.1.4 Deleting Files

To remove a file from the system, the user types the file specification followed by the /DE (delete) option. For example:

```
#TRY2.BAC/DE
```

This command string causes the file TRY2.BAC to be removed from the system disk if found under the current user's account number. No default assumptions are made as to the filename extension. An extension must be specified if the file was stored with an extension.

The user cannot delete a file under another account or a file which is write-protected against him. If an attempt is made to delete a non-existent file from a device, the error message ?CAN'T FIND FILE OR ACCOUNT is given.

More than one file can be deleted by specifying several file specifications, separated by commas. For example:

```
#FILE1.BAC,FILE2.BAC,DT1:FILE1.BAC/DE
```

deletes FILE1.BAC from both the public structure and DECtape unit 1 and deletes FILE2.BAC from the public structure. The number of files which can be deleted by one PIP command string is limited only by the maximum length of the command line.

### 17.1.5 Zeroing a Device Directory

Zeroing a device directory removes all files stored under one account number on the given device. In order to perform this operation, the user logs into the system under that account number (with the correct password) and runs PIP, giving the device specification followed by the /ZE (zero) option:

```
#/ZE
REALLY ZERO [120,80] SY:?
```

If the user does not type a device specification, PIP assumes the public structure and prints the REALLY ZERO account question.To cancel the operation, the user types the RETURN key. To remove all files from the current account on the system disk, he types Y or any string beginning with Y. Note that only the current account can be zeroed. The user must not specify an account number. The following example shows the command to zero a DECtape.

```
#DT1:/ZE
REALLY ZERO DT1:?
```

PIP prints the REALLY ZERO device question since no separation of files by account exists for DECtape. Hence, any user can remove all files from a DECtape reel by the zero action.

The extended version of PIP (described in Section 17.2) must be used to zero .ANSI formatted magtape.

### 17.1.6 Listing a Device Directory

The user can request a listing of all files under his account number, all files with a given name or extension under his account number, or a particular file under his account number on the public structure or any one or more devices. The format of the directory listing command is as follows:

```
#output=input file(s)/option
```

An output file specification can be supplied where an output device other than the user terminal is desired. The more usual form of the command is:

```
#input file(s)/option
```

in which case the directory listing is sent to the terminal issuing the command. Unless another device is specified, only a directory of the public structure is printed. For example, the sequence

```
RUN $PIP
PIP   V06B-03    RSTS V06B-02    TIMESHARING
#,DT0:,DT1:/DI
```

causes a full directory listing of files for the current user on the public structure (the blank device specification, indicated by the comma with no preceding characters indicates the public structure, and on DECtape units 0 and 1.

The options available with device listings are described in Table 17-4. The input file specifications are described in Table 17-5.

Table 17-4 PIP Directory Listing Options

| Option | Function |
|---|---|
| /BR | BRief directory listing is printed; includes only filenames and extension with four file specifications on each printed line. |
| /DI | Normal DIrectory listing is printed; includes filename, extension, length, protection code, and creation date. |
| /DI:S | Full (Slow) directory listing is printed; includes:<br>1. for disk devices: filename, extension, length, protection code, creation date and time.<br>2. for magtape or DECtape: filename, extension, length, protection code, creation date. |

Table 17-5 Input File Specifications

| Input File | Directory Printed Includes |
|---|---|
| none | all files on the public structure under the current user account number |
| dev: | all files under the current user account number on the device specified. |
| dev:*.* | same as the above. |
| dev:filename.* | all files having the filename specified, under the current user account number, on the device specified. |
| dev:name.ext | only the file specified, under the current user account number, on the device specified. |

Where no device is specified, the default device is the public structure; no message is printed if the particular input file specification(s) cannot be found; where several input file specifications are given, they are separated by commas. Account numbers are only significant for disk files and magtape files.

## 17.1.7 Guidelines for Transfer Operations and DECtape Usage

The PIP system program performs transfer operations in any one of several ways depending upon options the user specifies. The options are listed in Table 17-3.

If the user specifies no option in a transfer request, the system checks each byte of data for a CTRL/Z combination CHR$(26). PIP transfers the data block by block. The last block transferred by PIP is either the last block in the file or the block containing a CTRL/Z combination.

The ASCII type of transfer is convenient for moving BASIC-PLUS source files from one device to another. For example,

```
#DT1:FILNAM.BAS=FILNAM.BAS
#
```

PIP transfers the source code from the public structure to DECtape and terminates the transfer when the last block in the file is encountered. Any extraneous data following the END statement in the last block of the file causes no harm since the Run Time System recognizes the END statement as the end of file.

Under certain circumstances, PIP can be forced to discard unwanted NUL and RUBOUT characters. The user can specify the /FA option in the transfer request when he creates an ASCII text file from the keyboard or prints an ASCII text file at the keyboard. For example,

```
#KB:=FILNAM.TXT/FA

          (PIP prints the text.)

#
```

PIP terminates the transfer of the disk file upon encountering the CTRL/Z character. Any NUL or non-printing characters are discarded.

The ASCII transfer is not suitable for files containing 8-bit binary data. The bytes in a virtual core array or record I/O file can take any pattern of 8 bits, one of which can be that of the CTRL/Z character. If the user transfers such a binary file with the ASCII type of transfer request, the unpredictable occurrence of the CTRL/Z character pattern causes PIP to terminate the transfer and possibly loses data. To transfer such binary files between like devices, use the /BL option. For example,

```
#DT1:FILNAM.DAT=DT0:FILNAM.DAT/BL
#DK0:FILNAM.DAT=DK1:FILNAM.DAT/BL
#
```

PIP transfers the data between the devices strictly block by block and does not check any characters. The default block size of the device is used.

Since both magtape and disk have the same default block size (512 bytes), the user can also specify the /BL option in transfers between magtape and disk. For example,

```
#MT0:FILNAM.DAT=DK1:FILNAM.DAT/BL
#
```

PIP terminates the transfer when the system signals the end of file on the disk. The file on magtape contains 512 bytes per block.

Because the block sizes of DECtape (510 bytes) and disk (512 bytes) are different, PIP cannot properly handle block by block transfers of binary files between these two devices. If a DECtape block is transferred to disk, two extra bytes are generated on the disk; if a disk block is transferred to DECtape, two bytes are lost. Thus, to transfer binary files from disk to DECtape, the user must specify a special option – the /CO option. For example,

```
#DT0:FILNAM.DAT=DK0:FILNAM.DAT/CO
#
```

As a result of the user's specifying the /CO option, PIP transfers the file as a stream of contiguous bytes in 510 byte portions. In the last block on DECtape, the system fills any unused locations with NUL characters. In this manner PIP preserves the two extra bytes in the disk block. To transfer a binary file on DECtape to the disk, the user must specify the /CO:T option. For example,

```
#FILNAM.DAT=DT0:FILNAM.DAT/CO:T
#
```

As a result of the user's specifying the /CO:T option PIP transfers the file as a stream of contiguous bytes in 512 byte portions and truncates the last buffer. This action prevents the length of the file on disk from increasing when no new data is added. (A 10 block file on disk, for example, requires 10 blocks plus 20 bytes – a total of 11 blocks – on DECtape.)

Transferring a compiled file requires a special type of transfer. A compiled file is a unique form of a binary file. (Note that PIP allows only privileged users to transfer compiled files.) When transferring a compiled file, the user must specify one of the options described above and required for binary files. Moreover, if the output device is disk, the user can specify the /CL:4 option. This procedure forces PIP to create the output file in clusters of four contiguous blocks and is helpful since compiled files are always in 1K word clusters on the disk.

The user can transfer a file from a device whose block size is other than 512 bytes by specifying the /BL:n option. For example, to recall a file on magtape created with a block size of 2048 bytes, the user types a command similar to the following.

```
#FILNAM.DAT=MT1:FILNAM.DAT/BL:2048
#
```

PIP reads 2048 byte blocks from the magtape file and creates a disk file using a RECORD SIZE value of 2048.

The /BL:n option can also optimize transfers of large files from disk to disk or from disk to magtape. The advantage is gained by specifying a block size larger than 512. For example, if a large data file resides on a disk, the user can specify the /BL:2048 option in the transfer request. PIP subsequently reads four blocks at a time from the input file rather than executing four separate read operations and writes the output file using a RECORD SIZE value of 2048.

### 17.1.8 PIP as a CCL Command

The standard CCL definitions supplied by Digital include PIP as a CCL command. If PIP is included in the CCL for the system, the user may invoke the PIP program by typing:

```
PIP
```

This command produces the same response as

```
RUN $PIP
```

Also, typing

```
PIP<string>
```

causes PIP to run, process <string> as a standard PIP command, and return to the READY state. For example, if the following is typed in response to the READY message:

```
PIP LP:=FILE
```

a copy of FILE is printed on LP0:. Note that whatever was previously in memory before the PIP command was executed is now destroyed.

## 17.2 THE EXTENDED PIP PROGRAM

The extended PIP system program provides the following features:

1. accepting wild card characters in file name and extension specifications,
2. inspecting eligible files for transfer, rename, and delete operations,
3. pre-extending or extending an output file,
4. writing zeroes over data before a file is deleted,
5. reading DOS format disks for copy and directory listing operations,
6. handling ANSI format magtapes on transfer and directory listing operations,
7. processing indirect commands, and
8. continuing the command on the next physical line.

Extended PIP has all the features of the 8K version and includes additional features described in the following section. It requires a 16K word job area. To learn which version of PIP is available, the user types /HE while at PIP command level.

**NOTE**

For wild card processing in delete and rename commands, extended PIP uses a temporary file under the user's account to store directory information. If the user's account is full, PIP encounters the ?NO ROOM FOR USER ON DEVICE error when it attempts to open the temporary file. To overcome the error, the user issues the KILL system command to delete one file and then rerun PIP.

## 17.2.1 Wild Card Specifications

Extended PIP allows * and ? characters in file names and extensions to denote wild card elements and to thereby designate multiple files in a single file specification. For a description of wild cards, see Section 12.3.1.

In the absence of an element or elements of the full file specification, extended PIP substitutes defaults for file names or extensions or both. The following default interpretations are applied to the incomplete specifications shown.

| Specification | Default | Meaning |
|---|---|---|
| null | *.* | All files |
| * | *.* | All files |
| *. | *. | All files with null extensions |
| . | *. | All files with null extensions |
| .EXT | *.EXT | All files with extension EXT |
| FILE | FILE.* | All files with name FILE |

## 17.2.2 Extended PIP Defaults and Additional Options

Wild card characters allow multiple files to be designated in the input and output of a PIP command. Multiple file specifications can be given on input: each must be separated by a comma. Where multiple file specifications appear as input in a PIP command, the default interpretations shown in Table 17-6 apply for missing elements in each specification.

**Table 17-6 General Input Defaults**

| Missing Element | Default Interpretation |
|---|---|
| Account | The previously specified account. If missing from the first file specification given, the current user's account is the default. |
| Device | The previously specified device. If missing from the first file specification given, the public structure (SY:) is the default. |
| File name | The asterisk (*) specification |
| Dot and extension[1] | The asterisk (*) specification |

[1] If the dot is present, the extension is assumed to be present but can be null. When the dot is not present (and only then), the default extension is used.

On output, however, only one file specification is permitted although that file specification can designate multiple files. Only the asterisk (*) is permitted as a wild card character in an output specification. An attempt to use the question mark (?) in the output generates the ?INVALID DEVICE error. When the output is a wild card specification, the input specification is repeated for each file created as output.

Extended PIP applies specific default values for all elements in a transfer or directory listing command. The input and output defaults are summarized in Tables 17-7 and 17-8. The defaults for rename and delete commands differ from those applied to transfer commands. (Sections 17.2.4 and 17.2.5 describe the individual differences.)

**Table 17-7 Output Defaults for Transfer and Directory Commands**

| Element | Description | Default |
|---|---|---|
| dev: | device specification | The public structure (SY:) |
| [proj,prog] | project and programmer (account) number | Current user account number |
| name | file name specification | Input file name is used |
| .ext | file name extension | The input extension is used |
| <prot> | protection code | The system wide default (<60>), the default set by the ASSIGN command, or the value specified in the command. |
| null | no specification | The current keyboard (KB:PIPEXT.OUT) |

**Table 17-8 Input Defaults for Transfer and Directory Operations**

| Element | Description | Default |
|---|---|---|
| dev: | device specification | Immediately previous device specification. If none, then SY: is substituted. |
| [proj,prog] | project and programmer (account) number | Immediately previous account number. If none, then current user account number is substituted. |
| name | file name specification | All files (Extended PIP substitutes the * character) |
| | dot | All files with null extensions |
| .ext | file name extension | All extensions (Extended PIP substitutes the * character) |
| <prot> | protection code | The system wide default or the default set by the ASSIGN command. |
| null | no specification | All files on the immediately previous device and account. If none, SY: and current user account are used. |

The new options available in extended PIP are described in Table 17-9. Either the < or = character can separate the input and output sides of the PIP command.

**Table 17-9 Additional Extended PIP Options**

| Option | Function |
|---|---|
| /IN | Used in transfer, delete, and rename operations to inspect each file eligible for the operation. PIP prints the full specification of an eligible file and the ? character. To execute the operation for that file, YES (or any string beginning with Y) must be typed. To omit the operation for that file, NO (or any string not beginning with Y) can be typed.<br><br>The /IN option affects the individual file specification and not the entire command. For example,<br><br>`#A.*,B.*/IN,C.*/DE`<br><br>PIP deletes the files with names A and performs the inspect action for files with name B. It then deletes files with name C. |
| /PRX:n | Used on transfer operations to pre-extend a disk file to n blocks. If :n is omitted and the first input file is from disk or DECtape, this option pre-extends the output file to the length of the first input file.<br><br>Cannot be used with the /UP option. |
| /EX | Used on transfer operations to open the disk output file with MODE 2 (for appending data). (See Section 10.5.2 of the *BASIC-PLUS Language Manual.*) The output file is extended by appending the input file(s) to it.<br><br>Cannot be used with the /PRX, /CL, and /UP options. |
| /WO | Used on individual disk file specifications in delete commands. PIP writes zeroes in the file before it is deleted. The /WO option must appear with each file specification to which it applies, whereas the /DE option need appear only once in the command string. For example,<br><br>`#A.BAK/WO,B.BAK/DE`<br><br>The /DE option applies to both files but the /WO option applies only to the file A.BAK. |
| /VID:label | Required when using the /ZE option on ANSI standard magtape. The label is an alphanumeric volume label of the magtape to be zeroed. The following example shows the format of the command.<br><br>`#MT1:/VID:ABC/ZE` |
| /HE | Prints the help file PIP.TXT. /HE must be the only element in the input specification. |

(continued on next page)

**Table 17-9 (Cont.) Additional Extended PIP Options**

| Option | Function |
|---|---|
| /LI | Creates a directory listing.<br><br>For RSTS/E disk, DECtape, and DOS format disk, the following data is printed: name, extension, length (C denotes contiguous for DOS disk files), RSTS/E protection code, and creation date.<br><br>For DOS label magtape, the DOS protection code replaces the RSTS/E protection code and both the RSTS/E project-programmer number and the DOS user identification code are added to the listing.<br><br>For ANSI label magtape, the protection code is not applicable and is omitted. |
| /LI:S | Creates a full (slow) directory listing.<br><br>For RSTS/E disks only. Adds time of creation, data of last access, and file cluster size information to the standard directory listing.<br><br>For other devices, prints the standard directory listing. |
| /DOS | Indicates that the input disk is in DOS format. PIP can read the disk but cannot write on the disk. Thus, only transfer and directory operations are possible. |
| /BLOCK | Forces an ANSI magtape input file to be read as if it were written in ANSI:U format. |
| /FORMAT:U | Forces an ANSI magtape output file to have an undefined format (contiguous mode transfer required). If /BL:n is on the input file specification, the output file block length is n bytes. |
| /FORMAT:V | Forces an ANSI magtape output file to have a variable length record format. If /BL:n is on the input file specification, the output file block length is n bytes. |
| /MORE | Continues a command on the next physical line. |

### 17.2.3 Wild Card Specifications in Transfer and Directory Listing Commands

For transfer and directory listing requests in extended PIP, the standard defaults described in Tables 17-7 and 17-8 are applied. The sample commands and accompanying text in this section demonstrate the defaults applied to transfer requests. The same defaults also apply to directory listing requests. The following example illustrates the defaults applied to a multiple file transfer request.

```
#DK1:*.BAK=*.BAS
```

PIP transfers all files with .BAS extensions from the current account in the public structure. Each eligible file is created as a unique file on RK unit 1 under the current account. Each file has the same file name as the input file but is given an extension of .BAK and the default protection code.

The following command shows the device and account defaults used when elements are missing from the input.

```
#DK1:<60>=MT0:[2,2]*.MAC,*.BAK
```

PIP first transfers all files with .MAC extensions from account [2,2] on magtape unit 0 to the current user account on RK unit 1. In the absence of a device and account designation in the second input specification, PIP uses the immediately previous device and account as default. All files with .BAK extensions from account [2,2] on magtape unit 0 are transferred to the current user account on RK unit 1. The files are created with a protection code of <60>. (The magtape on unit 0 is rewound before the search for the .BAK files begins.)

The following command shows the default for a null input specification.

```
#MT0:[5,5]=DK0:/BL:1024
```

PIP transfers all files from the current user account in the public structure and all files from the current user account on RK unit 0. Each input file is created on magtape unit 0 with a project-programmer number of [5,5] and with the default protection code. The output records are written in 1024-byte blocks.

The following command shows the use of the /DOS option.

```
#DK1:=DT0:,DK0:[128,128]*.MAC/DOS,SY:*.BAS/FA
```

All files from DECtape unit 0, all files with .MAC extensions under UIC 200,200 (octal) on the DOS disk from RK unit 0, and all files with .BAS extensions under account [128,128] in the public structure are transferred to the RSTS/E disk on RK unit 1 with the default protection code. The /DOS switch applies only to RK unit 0; the UIC must be specified in decimal. Since no account specification appeared in the last file specification, the previous account specification is used. Files from DECtape unit 0 and from the public structure are transferred in formatted ASCII mode. Files from RK unit 0 are transferred in /CO mode because they come from a DOS disk.

The /IN option with a file specification causes PIP to list, one at a time as a query, the files eligible for transfer. This feature allows the user to inspect each eligible file and to selectively transfer each file according to the response given to the query.

The /IN option must appear with each specification for which the inspect feature is to apply. For example,

```
#*.*=DK0:[5,5]*.BAS/IN,*.SRT,*.CBL/IN/FA
```

In the command above, files with .BAS, .SRT, and .CBL extensions on RK unit 0 under account [5,5] are to be transferred to the public structure under the current user account. The inspect feature applies to files with .BAS and .CBL extensions because of the accompanying /IN option. To transfer an eligible file, type YES (or any string beginning with Y) in response to the inspect query. To omit an eligible file from the transfer, type NO (or any string not beginning with Y). Files with .SRT extensions are transferred without the inspect feature being applied.

### 17.2.4 Wild Card Specifications in Rename Commands

For rename commands, extended PIP does not apply the standard defaults in all cases. In the output, device and account defaults are taken from those applied in the input and the protection code is that of the input file. The device and account, therefore, need not be repeated in the output specification. For example,

```
#*.TMP=DK0:[5,5]*.BAS/RE
```

PIP renames all files with .BAS extensions in account [5,5] on RK unit 0. Each output file has the same name and protection code as the input file but is given the extension .TMP. No transfer occurs and only files on RK unit 0 are affected.

Except for protection code, the defaults in the input of a rename command are the same as those described in Table 17-8. The protection code of input files is preserved unless otherwise explicitly specified. The following example shows the device and account defaults applied on input.

```
#*.TMP=*.SRT,DK0:[5,5]*.SRT,DK1:*.SRT/RE
```

Extended PIP renames all files with .SRT extensions under the current account in the public structure; all files under account [5,5] on RK unit 0; and all files with .SRT extensions under account [5,5] on RK unit 1. The device and account defaults are applied whenever an explicit element is not given.

If the null specification is given as input, all files in the current account on the public structure are used. For example,

```
#.TMP=/RE
```

PIP renames all files and uses the name of the input file but gives the extension .TMP. If a file of the same name exists currently, PIP prints an error message but continues the rename operation.

The following command shows the defaults used when the protection code of files is changed.

```
#DK1:*.BAS<40>/RE
```

The protection code of all files with .BAS extensions under the current user account on RK unit 1 is changed to <40>. The original name and extensions are preserved because no name and extension is given in the output.

The /IN option in a rename command applies only to the file specification with which it appears. For example,

```
#*.BAK=DK1:*.BAS/IN,SY:*.BAS/RE
```

Only the eligible files under the current account on RK unit 1 are listed for inspection. The eligible file is renamed only if YES (or any string beginning with Y) is typed in response to the inspect query. If NO (or any string not beginning with Y) is typed, the file is not renamed. After all eligible files on RK unit 1 are processed, the eligible files under the current account on the public structure are renamed without the inspect action. If the inspect action is desired for the second specification also, the /IN option msut appear twice as shown below.

```
#DK1:*.BAS/IN,SY:*.BAS/IN/RE
```

As a result, the inspect action is applied to both specifications.

## 17.2.5 Wild Card Specifications in Delete Commands

For delete commands in extended PIP, device and account defaults are taken from the immediately previous device and account specified. If a device or an account is not explicitly specified, PIP applies the device and account defaults described in Table 17-8. The file name and extension must be explicit; extended PIP applies no defaults for either. The following commands show the proper procedure.

```
#DK1:[5,5]A.DAT,B.BAS/DE
```

The files A.DAT and B.BAS in account [5,5] on RK unit 1 are deleted. The device and account defaults are applied for the specification B.BAS. The following command shows the defaults applied in absence of an explicit device and account.

```
#A.DAT,B.BAS/DE
```

The files A.DAT and B.BAS are deleted from the current user account in the public structure. The following command shows the usage of the immediately previous device specification.

```
#ABC.*,DT1:ABC.*,*.ABC/DE
```

All files with name ABC are deleted from the public structure and from DECtape unit 1. For the third specification, PIP applies the immediately previous device default and deletes all files with extension .ABC on DECtape unit 1.

If the file name and extension are not explicit in a delete command, PIP prints an error message and terminates the operation. For example, the following commands generate errors.

| | |
|---|---|
| /DE | (file name and extension is missing) |
| .*/DE | (file name is missing) |
| */DE | (extension is missing) |

The /IN option in delete commands applies only to the file specification with which it appears. For example,

```
#DK1:*.BAS/IN,SY:*.BAS/DE
```

Only the eligible files under the current account on RK unit 1 are listed for inspection. The eligible file is deleted only if YES (or any string beginning with Y) is typed in response to the inspect query. If NO (or any string not beginning with Y) is typed, the file is not deleted. After all eligible files on RK unit 1 are processed, the eligible files under the current account on the public structure are deleted without the inspect action. If the inspect action is desired for the second specification also, the /IN option must appear twice as shown below.

```
#DK1:*.BAS/IN,SY:*.BAS/IN/DE
```

As a result, the inspect action is applied to both specifications.

**17.2.6 Processing ANSI Magtape Files**

Extended PIP handles ANSI D, ANSI U, and ANSI F formats on input and ANSI D and ANSI U formats on output. Table 17-10 shows the transfers possible.

**Table 17-10 Possible ANSI Magtape Transfers**

| Output | Input |
|---|---|
| ANSI D<br>ANSI U | ANSI D |
| ANSI D<br>ANSI U | ANSI U |
| ANSI D<br>ANSI U<br>ANSI F | ANSI F |

Unless otherwise specified, Extended PIP transfers a magtape file in the format in which it was written. For example,

```
#MT1:ABC.DAT=MT0:ABC.DAT
```

The input file from unit 0 is created on the output unit 1 in the same format in which it was written.

The format of the output file is altered if an option appears with the file specification. Again, the only changes in format allowed are those shown in Table 17-10. To force Extended PIP to read the input file as ANSI U, the /BLOCK option is used. The /BLOCK option forces the input file to be read as if it were written in ANSI U format. The option must appear with the file specification to which it applies. For example,

```
#SY:OUT=DK0:INP1.,MT0:INP2./BLOCK,MT1:INP3./BL:1024
```

The output file SY:OUT is written with a block size of 1024. The disk file INP1 is read with a block size of 1024. Because of the /BLOCK option, the magtape file INP2 is read as if it were written in ANSI U format with a maximum block size or 1024; that is, INP2 is read using GET statements without deblocking. The magtape file INP3, however, is read using the block size and record size with which it was written. If INP3 is in ANSI D format (variable record length), each record is deblocked to occupy one 1024-byte block on the disk. If INP3 is in ANSI U format (fixed record length), each block of input occupies one 1024-byte block on the disk. The /BLOCK option, therefore, causes files to be transferred more quickly since deblocking is not performed.

The /BLOCK applies only to the file with which it appears. For example,

```
#SY:OUT=DK0:INP1.,MT0:INP2./BLOCK/FA
```

The output file SY:OUT is written in formatted ASCII mode. The input file INP1 is read as a formatted ASCII file and INP2 is read as an ANSI U format file.

The /FORMAT:U and /FORMAT:V force the magtape output characteristics to the undefined format and the variable length record format, respectively. If neither switch appears, the files are processed according to their characteristics. For example,

```
#MT0:=MT1:
```

If the input unit is not assigned as ANSI, the output files are written in ANSI U format. If the input drive is assigned as ANSI, each output file is written in the same format as the corresponding input file. Specifying an output file name causes the input files to be merged into one output file having the characteristics of the first selected input file. Table 17-11 summarizes the effect of drive assignments in the above command.

**Table 17-11 Effect of Labeling Default Assignments**

| Input | Output | Effect |
|---|---|---|
| ANSI | ANSI | Output files are copied exactly from the inputvolume. All file header information is copied. |
| ANSI | DOS | Output files are created as if in a contiguous mode transfer. No file attributes are copied because DOS does not support file attributes. Input files are read in the format in which they were written. (For example, an ANSI D file is read as a variable length record file.) |
| DOS | ANSI | Output files are written in ANSI U format. |
| DOS | DOS | A /CO transfer is performed. |

(For information on assigning a labeling format to a magtape drive, see Section 20.2.3.)

To force an output file to have either undefined or variable length record format, the user specifies either the /FORMAT:U or the /FORMAT:V switch in the output.

## 17.2.7 Indirect Command Files in Extended PIP

Extended PIP can execute commands from a file. Such a file is termed an indirect command file and is indicated by the commercial at (@) character. The commands in the file are executed when an indirect command is given. An indirect command has the @ character as the first character on the command line followed by the file specification. The following is the correct format for an indirect command.

```
@dev: [proj,prog] file.ext
```

The defaults are the current account in the public structure and the extension .CMD. A file name must be specified in the command. The device can be either disk or DECtape.

The indirect command file is an ASCII file containing extended PIP commands. If a line of the file begins with a semi-colon (;) character, PIP treats it as a comment line and skips to the next line of the file. The REALLY ZERO and inspect questions of the /ZE and /IN options are not printed when executed from an indirect command file.

An indirect command can appear in the indirect command file. Ten levels of nesting are allowed. If more than 10 levels are attempted, the program prints the message INDIRECT COMMAND ERROR – COMMAND STACK OVERFLOW.

Most errors cause PIP to return to the base level – the level at which the first indirect command file was initiated. On the following errors, however, PIP simply prints the error message and continues processing.

```
NO FILES MATCHING <specification>     for /DE operations
NO FILES MATCHING <specification>     for /LI operations
```

where specification is the full file specification

```
?FILE OR ACCOUNT ALREADY EXISTS for /RE operations
```

The following command shows the procedure to execute an indirect command file.

```
#@ABC
```

PIP reads the file named ABC.CMD in the current user account in the public structure. If only the designator KBn: appears in the indirect command, PIP reads the specified keyboard for commands. Execution of indirect commands from the keyboard is terminated when the CTRL/Z combination is typed.

Since the logically assigned account (see Section 5.7) and the PIP indirect command are denoted in the same manner, the placement of the @ character on the PIP command line is important. For example, to obtain a directory listing of the logically assigned account, the @ character must be used with the /LI option. Because the @ character must be followed by a file name, the command @/LI generates an error. The following command, however, executes properly.

```
#SY:@/LI
```

The directory of the logically assigned account in the public structure is generated.

### 17.2.8 Extending the Physical Command Line – /MORE

The option /MORE at the end of a command line causes PIP to print the text MORE > and space to the next tab position. The command line can thereby be continued on the next physical line. The entire command is not executed until /MORE does not appear in the command line. For example,

```
#LP0:=A.DAT,B.BAS,C.BAT,/MORE

MORE > D.BAK/LI
```

PIP executes the /LI option for the files A.DAT, B.BAS, C.BAT, and D.BAK.

## 17.3 COPYING BETWEEN DEVICES: THE COPY PROGRAM

All of the information on a DECtape, magtape or disk can be copied by using the COPY system program and the /FC (fast copy) option. COPY is called as follows:

```
RUN  $COPY
```

COPY prints a pound sign (#) when it is ready to receive instructions. To print a help message, the user types /HE. To copy a device, the user must respond in the format shown below.

```
#new  device=old  device/FC
```

For example:

```
#DT1:=DT2:/FC
```

In this example, all of the data, programs and directories are copied, block by block, from the DECtape on unit 2 to the DECtape on unit 1. The original information on unit 2 is, of course, still intact.

Notice that all of the information on a device is copied; individual files cannot be specified.

Magtapes, DECtapes, disk cartridges and disk packs can be copied. In addition, information can be copied only to different units of the same device. That is, COPY can be used to copy a DECtape to another DECtape or a magtape to another magtape, but not to copy, say, a DECtape to magtape or vice versa. If an attempt is made to copy the information on one type of device onto another type of device, the error message MUST HAVE SAME TYPE DEVICES is given.

Finally, if the same unit number of a device is specified twice, the error message MUST HAVE DIFFERENT UNIT NUMBERS is given.

When the information is successfully copied from one device to another, the program terminates and the system prints READY.

To verify that the information on a device unit has been copied properly, the user issues the /VE (verify) option to the COPY program in either of the following ways:

```
#DT1:=DT2:/FC/VE
```

or

```
#DT1:=DT2:/NC/VE
```

Since verification is performed block by block, it requires as much time as the copy operation. Therefore, it is important to understand the difference between the above commands. The first command shown above, which includes the /FC option, copies information from DT2 to DT1 and then verifies that the information was copied correctly. The second command, which must include the /NC (no copy) option, does not copy information; it only verifies that the information on both DECtapes is identical.

The two sample commands shown above are the only allowable forms for verifying. If the user types a command string omitting the /FC or /NC option, the error message /FC OR /NC MUST BE SPECIFIED is returned.

If the user specifies both the /FC and /NC options in the same command line, the error message CANNOT SPECIFY BOTH /FC AND /NC is returned.

As with the /FC option, individual files cannot be specified with the /VE option; all of the information on a device is verified.

As the verification is performed, the first message COPY prints is:

```
BEGINNING VERIFICATION PASS
```

If all of the information has been copied correctly from one device to another, the next message COPY prints is:

```
VERIFICATION COMPLETE 0 BAD BLOCKS

READY
```

If, however, the information has not been copied correctly, COPY prints the decimal number of blocks in which inconsistencies appear. For example,

```
#DT1:=DT2:/FC/VE
BEGINNING VERIFICATION PASS
THE FOLLOWING BLOCKS ARE BAD:
17
31
89

VERIFICATION COMPLETE 3 BAD BLOCKS

READY
```

The /BL option, which specifies blocksize, speeds up copying or copies magtapes written with non-standard record sizes. To speed up copying, the user should specify a larger blocksize. For example,

```
#DK0:=DK1:/BL:2048/FC
```

causes the disk on drive DK1: to be copied to the disk on drive DK0: in 2048-byte blocks, rather than the default 512-byte blocks. If the user specifies an illegal block size, the error message ILLEGAL BLOCK SIZE is returned.

To specify other than standard density and parity settings when copying magtape, the user issues the /DENSITY:d and /PARITY:p options. These may be minimally abbreviated to /DE:d and /PA:p. In these options,

| d can be: | p can be |
|---|---|
| 200 | ODD |
| 556 | EVEN |
| 800 | |
| DUMP | |
| 1600 (phase-encoded) | |

If the user specifies a density or a parity that does not appear in this list, the error message ERROR IN SPECIFYING DENSITY (or PARITY) is returned.

The following example illustrates /DENSITY and /PARITY:

```
#MT0:/DENSITY:556/PARITY:EVEN <MT1:/DENSITY:DUMP/FC
```

COPY reads magtape unit 1 at 800 bits per inch dump mode and writes unit 0 in even parity at 556 bpi.

Several error messages are returned for general command errors. If the user misplaces or mistypes an option, the program prints ERROR IN SPECIFYING SWITCHES. If the user specifies an illegal or nonexistent device, the program prints ILLEGAL DEVICE. If he includes anything besides a device in the input or output specification, the program prints ILLEGAL INPUT (or OUTPUT) SPECIFICATION. Finally, if he types a format that the program cannot decode, it prints SYNTAX ERROR IN COMMAND STRING.

# CHAPTER 18
# STORING FILES OFF-LINE: THE BACKUP PROGRAM

## 18.1 PRESERVING FILES: THE BACKUP PROGRAM

The BACKUP system program allows the user to preserve off-line file copies on disk or on magtape. BACKUP, in its RESTORE mode, can also return those off-line copies to the on-line disk structure. BACKUP communicates with the user by dialogue: the program asks the user a sequence of questions which the user answers one at a time. His answers determine the program's mode, the device to which files will be copied, which files will be copied, and where BACKUP will print its directory and command listings: the record and results of its work.

The group of disks or magtapes on which the files are preserved is called the backup set; a single member of this set, whether a disk or a magtape, is called a volume.

Once BACKUP has accepted the user's responses in the dialogue, it selects the files to be processed, recording the accounts to be transferred and sorting them in ascending numerical order. Information about accounts, files, and errors is written into the work-file, which BACKUP uses for communication between modules and for writing its primary index file. Mainly a directory of the backup set, this file is written in duplicate on the set's last volume, called the index volume. Though the primary index file contains a complete directory, it is not in RSTS/E format. To create a permanent index file in RSTS/E format, the user specifies an extension other than the default .TMP. The resulting final index is called the auxiliary index file.

In transferring the files, BACKUP copies them to the backup medium. After it transfers all files, it updates the listing file with an entry for each account and file it transfers and for each error it encounters. Also, in a summary line for each account, BACKUP reports the number of files and blocks it has transferred and the number of errors it has encountered. When a backup volume is filled, BACKUP requests a new volume by printing a prompt on the job's keyboard. When the files have been transferred, BACKUP prints a message of completion and returns to the user's default run-time system.

**NOTE**

In order to back up files to a disk, a non-privileged user must have a disk in BACKUP format. This format is not the format of RSTS/E. To change a RSTS/E disk to BACKUP format, the non-privileged user must ask the system manager or a privileged user to run the program BACDSK.

### 18.1.1 The RESTORE Mode of BACKUP

In RESTORE mode, BACKUP performs a transfer the opposite of that just described: it transfers files from the backup set to a RSTS/E format disk. Like the primary BACKUP mode, RESTORE uses a dialogue to obtain information from the user.

In this dialogue, the user specifies the files for transfer – the entire backup set or a subset of it. He may also enter files by account, and exclude files from the transfer if they already exist on the destination disk.

RESTORE mode, in selecting the files for transfer, copies account and file information from the user-specified index file to a temporary disk file. On completing this process, the program, as it does in BACKUP mode, records in the listing file the number of accounts, files, and blocks to be transferred, along with accounting statistics.

Once this selection and listing of files is complete, the program, in RESTORE mode, copies the files from the backup set to the destination disk. Files and accounts are processed in their order of appearance on the backup volume. For each account entered and transferred, and for each file transferred, the program generates an entry in the listing file. Also logged in the listing file is any error that may destroy data. Such an error is printed on the job's keyboard as well. When all files and accounts have been transferred and all pertinent information logged, the program prints a completion message and returns to the user's default run-time system.

**NOTE**

RESTORE mode does not supersede files that are open for update. If an open file is encountered during transfer, the program generates an error message.

## 18.2 RUNNING BACKUP

To run the BACKUP program, the user types

```
RUN $BACKUP
```

This command starts the dialogue. BACKUP's first query is

```
BACKUP OR RESTORE?
```

To this query, the user may type, minimally, BAC or RES. BAC (or BACK, BACKU, etc.) starts the dialogue for a BACKUP operation, while RES (or REST, etc.) starts the dialogue for a RESTORE operation.

**NOTE**

BACKUP cannot be run under the control of the Batch processor.

### 18.2.1 File Specifications as Dialogue Answers

Several questions in the dialogue call for file specifications as answers. The user must include in these specifications certain characteristics of the files before BACKUP will process them. The characteristics may include filename, account, and dates of creation and last access. The user may also use this information to specify files as exceptions from a process.

The BACKUP file specification takes the following form:

```
filename/keyword:comparison:date
```

The filename is in RSTS/E format and may contain the wild-cards * and ?. The keyword is CREATION or ACCESS. (Both may be abbreviated to their first two letters.) The comparison is BEFORE or AFTER. (Both may be minimally abbreviated to their first three letters.) The date is in dd-mmm-yy format; if the user omits yy, BACKUP assumes the current year. A file specification entered in the BACKUP dialogue, therefore, might look like this:

```
*.TEC/AC:BEF:06-JUN-76
```

In this example, the filename *.TEC designates all files on the default account with the extension .TEC. The rest of the specification – the keyword AC[CESS] followed by the comparison BEF[ORE] and the date – narrows the category to only those .TEC files accessed before 06-JUN-76 (i.e., 05-JUN-76 or earlier).

A user may also specify the time of day in the creation field:

```
PHONE.*/CR:AFT:07-MAR-76:21:13
```

This specification designates all files with name PHONE that have been created since 21:13 (9:13 p.m.) on 07-MAR-76.

Only one creation phrase and one access phrase are allowed in each file specification. These phrases must be separated from each other and from the filename (even if it is null) by a slash (/). A specification applied to the default filename, therefore, might look like this:

```
/CR:BEF:01-APR-76:17:30/AC:AFT:01-SEP-76
```

The specification designates all files with the default filename that were created before 17:30 (5:30 p.m.) on 01-APR-76 and accessed after 01-SEP-76. Note that the components of each phrase are separated by a colon (:).

The user may exempt one or more files from an operation by including an EXCEPT phrase in the file specification. EXCEPT (which may be minimally abbreviated to EXC) is separated by a slash from other phrases and has two forms:

1. EXCEPT:file specification
2. EXCEPT:(list of file specifications)

The user specifies the first form if he wishes to exempt only one file specification and if this specification does not include a creation or access phrase. For example, the specification

```
[100,250] EDIT.*/CR:AFT:01-JAN-76/EXC:EDIT.TMP
```

designates all files on account [100,250] with name EDIT – except EDIT.TMP – that were created after 01-JAN-76.

Should the user wish to exempt more than one file specification or to include a CREATION or ACCESS phrase in the exception, he employs the second form. The entire specification might look like this:

```
*.*/CR:BEF:29-FEB-76/EXC:(*.RNO/CR:AFT:01-FEB-76,MT?.*/AC:AFT:31-DEC-75)
```

This specification initially designates all files on the default account that were created before 29-FEB-76. The exceptions, however, are:

1. All files with extension .RNO that were created after 01-FEB-76, and
2. Any file whose name begins with MT and is three characters long that has been accessed since 31-DEC-75.

An EXCEPT phrase, unless it has its own CREATION or ACCESS phrase, will be affected by the CREATION or ACCESS phrase already in the specification. A specification may include only one EXCEPT phrase.

For a summary of the BACKUP file specification, see the *RSTS/E Pocket Guide.*

### 18.2.2 Typographical Considerations

1. Continuing a line: A list of file specifications may be long enough to require several lines. BACKUP, therefore, accepts a hyphen (-) as the last character before a line terminator to indicate that the next line is a continuation of the current line. After the user types the hyphen and line terminator, BACKUP prompts with CONT>. BACKUP does not process any responses until the user has entered all continuation lines.
2. Spacing (blanks): In specification lines, BACKUP ignores blanks as separators, but reads them as terminators. The specification EXC EPT:BEE.BAS is illegal because the scan ends after EXC. EXCEPT : BEE.BAS, however, will be processed.
3. Making a comment: To place a comment in a command line, the user types an exclamation point (following any continuation hyphen), then the comment. Since ambiguity may result if the user also specifies account [1,3] as !, he must enclose that account specifier in quotes ("!" or '!'). These quotes will be stripped from the command line when it is processed.

### 18.2.3 Rules of the Dialogue

If there is a default answer to one of BACKUP's prompting questions, it will appear in angle brackets and before the question mark. In this example, the default is the user's keyboard:

```
LISTING FILE <KB:>?
```

By making a null response (i.e., simply pressing the RETURN key), the user receives the default. If he makes a null response to a question that has no default, BACKUP prints the message NO DEFAULT and reprints the question.

The user may request BACKUP's assistance with any question by typing /HELP.

Should BACKUP encounter a syntax error, it prints the appropriate error message and repeats the prompting question, preceding it with a question mark. The user, by responding with a question mark and pressing RETURN, can cause BACKUP to reprint the erroneous line up to the point of the error, and then to repeat the prompting question.

## 18.3 THE DIALOGUE

BACKUP starts the dialogue with the question

```
BAC[KUP] OR RES[TORE] ?
```

The user responds with BAC or RES (minimally) to specify his chosen mode. If, on the other hand, he wishes to see the help text for the whole program, he may type /HELP instead.

If the user wishes to create an indirect command file containing his responses to the dialogue, he may so specify by typing the option /SAVE (or, minimally, /SA) after his choice of mode, in the format:

```
BAC/SA
```

or

```
RES/SA
```

To the /SAVE option, BACKUP will respond with the question

```
INDIRECT FILE NAME<SY: [CUR ACT] BACKUP.CMD>?
```

or

```
INDIRECT FILE NAME<SY: [CUR ACT] RESTOR.CMD>?
```

depending on whether the user has chosen BACKUP or RESTORE mode. As the defaults – in angle brackets – indicate, a null response will specify the command file as BACKUP.CMD or RESTOR.CMD, on the system disk and under the current account. In specifying an indirect command file other than the default, the user may designate a disk or DECtape. The device must be accessible to the user. Into this file, BACKUP will print the rest of the responses.

The rest of the dialogue depends, of course, on the mode chosen in response to the first question. Described first is the dialogue for a BACKUP operation, run when the user responds with BAC.

**Table 18-1 Backup Dialogue Summary**

| Question | Response | Meaning |
|---|---|---|
| 1. BAC[KUP] OR RES[TORE]? | | |
| | BAC[KUP] (/SA[VE]) | Perform a Backup. |
| 1a. INDIRECT FILE NAME <SY: [CUR ACT] BACKUP.CMD>? | | |
| | File spec | Create an indirect command file with the specified name. Valid output devices are disk, and DECtape. Asked only if the /SAVE option is appended in Question 1. |
| 2. WORK-FILE NAME <SY: [CUR ACT] BACKnn.TMP>? | | |
| | File spec | Use the specified file as the work-file. If the default is used, nn is the job number. Any disk may be substituted for SY:. (This file can be used as the auxiliary index file.) |
| 3. LISTING FILE <KB:>? | | |
| | KBn:, LPn: | Write the listing file to the device specified. |
| | File spec | Write the listing file to the specified file on disk or DECtape. The default file name is [CUR ACT] BACKUP.LST. |
| 4. FROM DISK <SY:>? | | |
| | Disk name | Back up from the specified disk. This disk must remain mounted throughout the entire Backup process. |
| 5. FROM FILE(S) < [CUR ACT] *.*>? | | |
| | File spec(s) | Back up the specified files. |
| 6. TO DEVICE <MT:>? | | |
| | MT:, disk name | Back up to the specified medium. |
| 7. BEGIN AT < [*,*] *.*>? | | |
| | File spec | Back up starting with the file specified. The default starts with the first file matching the file specification in Question 5. Answer with a single file specification only. No EXCEPT phrases are allowed. |
| 8. DELETE FILE(S)<NONE>? | | |
| | File spec(s) | Delete the specified files after backing them up. |
| 9. COMPARE FILE(S) <NONE>? | | |
| | File spec(s) | Compare the specified files to the originals after backing them up. |

### 18.3.1 The BAC Dialogue

In the following description of the dialogue, BACKUP's questions are underlined and followed by explanations.

WORK-FILE NAME <SY: [CUR ACT] BACKnn.TMP>?

This question asks the user to specify the work-file, a record of accounts, files, and errors that BACKUP uses to create its primary index file. The user may later copy the work-file with PIP to create the auxiliary index file. As the default indicates, a null response will specify the work-file as BACKnn.TMP (where nn is the current job number), under the current account and on the system disk. In place of the default, the user may choose any file specification, provided that the file is on disk. If the user is creating a large backup set, it is recommended that he specify a private disk for the work-file in order to accommodate its length.[1] It is also recommended, for the security of the work-file, that he copy and rename it with PIP after completing the backup and before logging out, since on some systems a .TMP file is deleted on logout.

**NOTE**

An easy way to safeguard the work-file is to specify an extension other than TMP in response to the WORK-FILE NAME question. If the user performs this action, he need not copy the work-file with PIP.

In order to save space, the user may wish to prevent the work-file from being transferred to the backup volume. The way to prevent this transfer is to specify the work-file in an /EXCEPT phrase as part of the response to the FROM FILE(S) question.

LISTING FILE<KB:>?

This question asks where the user wishes BACKUP to print the listing file BACKUP.LST, a record of files and blocks transferred and errors encountered. As the default <KB:> indicates, a null response will cause BACKUP.LST to be printed at the user's keyboard. To request printing at any other valid, available keyboard or other non-file structured device, the user may specify a device designator. Thus, if he responds with KB0: the listing file will have the specification

```
KB0: [cur act] BACKUP.LST
```

Instead of a non-file structured device, the user may give any file specification. The default specification string applied to the response is

```
SY: [CUR ACT] BACKUP.LST
```

The listing file will be opened for output and accessed immediately, in order to verify that the file/device is available. Furthermore, the device will be ASSIGNed immediately, and will remain ASSIGNed for the duration of the run.

FROM DISK<SY:>?

This question asks the user to specify the input device, from which the files will be transferred; the default is SY:. Only one device, which must be a disk, may be specified. Filenames, extensions, and accounts are to be specified in response to the next question.

FROM FILE(S)< [CUR ACT] *.*>?

---

[1] As a rule of thumb, the following formula may be used to calculate the size of the work file in blocks:

.15 * (ACC + FIL + ATT)

where ACC is the number of accounts, FIL the number of files, and ATT the number of files with attributes.

This question asks the user to specify the file(s) he has chosen to back up from the input disk(s). The default is all files from the current account. In entering specifications other than the default, the user follows the format described in Section 18.2.1. Multiple specifications must be separated by commas. The default for each specification is all files on the current account. BACKUP transfers the files in the response according to the following rules:

1. Any file to which the user has read access (except 0 length files) will be transferred.
2. Files are transferred in the order in which they exist in the directory, not in the order in which they are specified.
3. Accounts are transferred in numerically increasing order, regardless of their order in the MFD.
4. If SY: is the input device, and the public structure contains two or more disk devices, files from the same account on all public disks are transferred together before the next account is processed. Furthermore, only accounts which exist in the MFD of the system disk are processed.

TO DEVICE<MT:>?

This question asks the user to specify the type of device to which the files specified in the previous response will be transferred, or "backed up." The default device is magtape; the alternative is disk (DK:, DP:, DB:). Only one device specification may be given, and no unit number may be specified.

BEGIN AT <[*,*] *.*>?

This question asks the user to specify the first file to be transferred. The default is the first file matching the user's response to the FROM FILE(S) question. The user should note that he can respond only with a specification for a single file. The default for any specification entered is the first matching filename in the FROM FILE(S) response and on the current account. All files and accounts occurring before the file specified will be skipped – i.e., will not be transferred. Because of this fact, the user may give a general response to the FROM FILES question, or may accept the default (all files from the current account), rather than specifying multiple files. Thus, the user saves some typing time. Note that no /EXCEPT phrase is permitted in a BEGIN AT specification.

DELETE FILE(S) <NONE>?

This question asks the user to specify any file(s) he wishes to be deleted from the input disk after they have been backed up. As the default NONE indicates, a null response will preserve all input disk files. But if the user wishes a file or set of files to be deleted from the input disk, he indicates them by a specification or set of specifications (separated by commas) in the format described in Section 18.2.1. There is no default specification string; all fields, including account number, must be specified.

Not all input disk files matching the DELETE specifications will be deleted – only those which are selected and transferred. Moreover, BACKUP will not delete a file to which the user lacks write access, or a file which caused an error during transfer or verification.

COMPARE FILE(S)<NONE>?

This question asks the user to specify any backed-up files he wishes to be checked against their originals on the input disk. As the default <NONE> indicates, a null response means that no comparison will be performed. But if the user wishes BACKUP to compare a file or set of files. he enters a specification or set of specifications (separated by commas) in the format described in Section 18.2.1. Note that in the actual transfer, this comparison occurs before any deletions.

**Table 18-2 Restore Dialogue Summary**

| Question | Response | Meaning |
|---|---|---|
| 1. BAC[KUP] OR RES[TORE]? | | |
| | RES[TORE] (/SA[VE]) | Perform a Restore. |
| 1a. INDIRECT FILE NAME <SY: [CUR ACT] RESTOR.CMD>? | | |
| | File spec | Create an indirect command file with the specified name. Valid output devices are disk, DECtape, and terminal. Asked only if the /SAVE option is appended in Question 1. |
| 2. WORK-FILE NAME <SY: [CUR ACT] RESTnn.TMP>? | | |
| | File spec | Use the specified file as the work-file. If the default is used, nn is the job number. Any disk may be substituted for SY:. |
| 3. LISTING FILE <KB:>? | | |
| | KBn:, LPn: | Output the listing file to the specified device. |
| | File spec | Write the listing file to the specified file on disk or DECtape. The default file name is [CUR ACT] RESTOR.LST. |
| 4. FROM DEVICE <MT:>? | | |
| | MT:, disk name | Restore backed up files from the specified medium. |
| 5. INDEX FILE <PRIMARY>? | | |
| | File spec | Use the specified file as the index file. If the user accepts the default, Restore uses the Primary index file, which is on the final volume of the backup set. |
| 6. FROM FILE(S) < [CUR ACT] *.*>? | | |
| | File spec(s) | Restore the specified files. |
| 7. TO DISK <SY: [*.*] >? | | |
| | Disk name | Restore the specified files to the designated disk. |
| 8. BEGIN AT < [*,*] *.*>? | | |
| | File spec | Restore starting with the file specified here. The default starts with the first file matching the file specification in Question 6. The answer must be a single file specification. No EXCEPT phrases are allowed. |
| 9. SUPERSEDE <NONE>? | | |
| | File spec(s) | Overwrite the specified files on the destination disk with the backup versions. |
| 10. COMPARE FILE(S) <NONE>? | | |
| | File spec(s) | Compare the restored files to the backup versions. |

### 18.3.2 The RES Dialogue

In the following description of the dialogue, BACKUP's RESTORE mode questions are underlined and followed by explanations.

WORK-FILE NAME <SY: [CUR ACT] RESTnn.TMP>?

This question asks the user to specify the work-file. The default, specified by a null response, is RESTnn.TMP (where nn=current job number), under the current account and on the system disk. The user may choose any disk file specification in place of the default.

LISTING FILE <KB:>?

This question asks where the user wishes BACKUP to print RESTOR.LST. The default, specified by a null response, is the user's keyboard. To request printing at any other valid, available keyboard or other non-file structured device, the user may specify a device designator. Thus, the response LP0: will specify the following listing file:

```
LP0: [CUR ACT] RESTOR.LST
```

Instead of a keyboard or other non-file structured device, the user may give any accessible file specification. The default specification string applied to the response is

```
SY: [CUR ACT] RESTOR.LST
```

FROM DEVICE <MT:>?

This question asks the user from what type of device the backed-up files will be restored. The default, specified by a null response, is a magtape; the alternate response is a disk.

INDEX FILE <PRIMARY>?

This question asks the user to specify the index file. In RESTORE mode, BACKUP uses this file as a source for information about the accounts and files selected for restoration. From this information, BACKUP writes the work-file. When BACKUP has completed the restoration, it will copy this information into the listing file RESTOR.LST.

As the default indicates, a null response will cause BACKUP to use the primary index file as a source.

If the user specifies an index file other than the default, BACKUP will load its work-file from user-specified information in that index file. The user may specify a disk or DECtape; for example,

```
DT1:MYRES.IND
```

FROM FILE(S)< [CUR ACT] *.*>?

This question asks the user to specify the backed-up file(s) to be restored to the destination disk. The default is all backed-up files from the current account. In entering specifications, the user follows the format in Section 18.2.1. The account for each specification is the one last given in the response, including the default account specified by a null. For rules and restrictions governing the transfer, see the description of this question in BAC mode (Section 18.3.1).

TO DISK <SY: [*,*] >?

This question asks the user to specify the destination disk and account to which the backed-up files should be transferred. A null response will cause BACKUP to transfer all files to the system disk on the accounts from which they were backed up. This default action applies also to any specification entered by the user. Note that a user cannot restore a file to an account to which he does not have access. This restriction holds true even if that file was created on the inaccessible account.

BEGIN AT <[*,*] *.*>?

This question asks the user to specify the first file to be transferred. The default is the first file matching the user's response to the FROM FILE(S) question. The user should note that he can respond only with a specification for a single file. The default for any specification entered is the first matching filename in the FROM FILE(S) response and on the current account. No /EXCEPT phrases are allowed.

SUPERSEDE <NONE>?

This question asks the user to specify a file or set of files that he wishes BACKUP to overwrite. The RESTORE mode, in transferring the backed-up files he specifies, will cause them to replace their originals on the destination disk. If the user makes a null response, no files will be overwritten; i.e., if they already exist on the destination disk their copies on the backup device will not be transferred.

The user, in responding with a specification or set of specifications, follows the format in Section 18.2.1. BACKUP matches these specifications against file information stored on the backup device, not in the directory of the destination disk.

**CAUTION**

A user responding to SUPERSEDE should take care that the response does not unintentionally replace a recently created disk file with an older one from the backup set.

COMPARE FILE(S) <NONE>?

This question asks the user to specify any file(s) transferred to the destination disk that he wishes to be checked against their copies on the backup device. A null response causes no such comparison. But if the user wishes BACKUP to compare a file or set of files, he specifies the file(s) according to the format in Section 18.2.1.

## 18.4 INTERRUPTION COMMANDS

BACKUP provides the non-privileged user with six commands that he may issue during transfer phase, while the program is processing files. Two of these commands may also be issued during the earlier select phase, while the program is identifying the files it will process. These six interruption commands can terminate, suspend, continue, and report on BACKUP's processing. To show its readiness to accept an interruption command, BACKUP prints an asterisk (*) on the job's keyboard. The user may issue a command any time after this asterisk appears. Table 18-3 presents the interruption commands, the phases in which they will work, and their functions.

## 18.5 RUNNING BACKUP FROM AN INDIRECT COMMAND FILE

The user may run BACKUP from an indirect command file created by specifying the /SAVE option described in Section 18.3. When BACKUP runs from such a file, it does not print dialogue questions; they are unnecessary because the program reads the user's responses directly from the file. BACKUP does, however, print an asterisk (*) to show its readiness to accept interruption commands. And, when the transfer is complete, BACKUP prints a completion message.

To run the program, in BACKUP or RESTORE mode, from an indirect command file, the user responds to the first question – BAC[KUP] OR RES[TORE]? – by typing a commercial 'at' sign (@) followed by the filename. For example, assume that the user, in exercising the /SAVE option during a previous run, accepted the default filename for the indirect command file. His response would then be

```
    BACK @BACKUP.CMD
```

**Table 18-3 Interruption Commands**

| Command | Phase | Function |
|---|---|---|
| ABORT, ABO | Both (Select, Transfer) | Terminates processing immediately and returns to user's default run-time system. |
| CONTINUE, CON | Both | Continues processing after a PAUSE.. |
| END | Transfer | Terminates processing after the current file has been transferred. |
| PAUSE, PAU | Both | Suspends execution until user types CONTINUE. During a PAUSE, any other legal command works. But note that ABORT will supersede the pause and will cause an exit from BACKUP. |
| STATUS, STA | Both | Prints, on KB:, a backup status report containing processed accounts, files, and blocks, and the current account and file; the information depends on the phase. |
| TERMINATE, TER | Transfer | Closes the current volume at the end of the current file or of the current output volume, whichever comes first (not valid during a RESTORE operation). |

or

```
RES @RESTOR.CMD
```

But if the user specified an indirect filename of his own choice, MYBAC.CMD for example, his response would be

```
BAC @MYBAC.CMD
```

Note also that if the assignable account specifier is used, two commercial 'at' signs are necessary, as in the following example:

```
BACKU @@MYBAC.CMD
```

## 18.6 TWO DIALOGUE EXAMPLES: BAC AND RES

The first example is of a BACKUP mode dialogue in which the user chooses all allowable defaults. The user's input is underlined.

```
BAC[KUP] OR RES[TORE] ? BAC

WORK-FILE NAME <SY:[10,25]BACK15.TMP>?

LISTING FILE <KB:>?
```

```
FROM DISK <SY:>?

FROM FILE(S) < [10,25] *.*>?

TO DEVICE? <MT:>?

BEGIN AT < [*,*] *.*>?

DELETE FILE(S) <NONE>?

COMPARE FILE(S) <NONE>?
```

The user is logged in under account [10,25] ; the job number is 15. By default, the work-file will be named BACK15.TMP, and the listing file will appear on the user's keyboard. Also by default, the files to be backed up are from his account on the public structure. By default, BACKUP will begin the transfer with the first file on his account, and will not delete any files; nor will it compare the backed-up files with their originals on the public structure.

The second example is of a RES mode dialogue in which the user does not accept all the defaults, but enters some responses. The user's input is underlined.

```
BAC[KUP] OR RES[TORE] ? RESTOR

WORK-FILE NAME <SY: [10,25] REST22.TMP>?

LISTING FILE <KB:>? TODAY.LST

FROM DEVICE <MT:>?

INDEX FILE <PRIMARY>? DT1:MYRES.IND

FROM FILE(S) < [10,25] *.*>? [10,*] *.LST

TO DISK <SY: [10,25] >? DK0:

BEGIN AT < [*,*] *.*>

SUPERSEDE <NONE>?

COMPARE FILE(S) <NONE>?
```

The user is logged in under account [10,25] ; the job number is 22. By default, the work-file will be named REST22.TMP. The listing file, named TODAY.LST by the user's choice, will appear on his keyboard by default. The backed-up files to be transferred are on magtape. The user specifies the index file as MYRES.IND, on DT1:. The backed-up files to be restored all have extension .LST, are under accounts [10,*], and are read-accessible to the user. These files will be restored onto the disk on DK0: (by the user's choice). By default, transfer will begin with the first file matching the user's response to the FROM FILE(S) question – i.e., the first .LST file found in account [10,*].

If a file of the same name already exists on the DK0: disk, it will not be restored to that disk, because the user accepts the default response <NONE> to the question SUPERSEDE. Finally, he accepts this default response to the question COMPARE as well, so that BACKUP will not check the restored files against their backed-up copies on MT0:.

## 18.7 MOUNTING AND DISMOUNTING VOLUMES

After BACKUP selects all files and accounts for transfer, it prints on the job's console terminal requests for labeling information and the device unit number for the first backup volume. It also prints a volume identification summary.

The labeling information BACKUP requests is the name and expiration date for the backup set. BACKUP uses the name as an identifier for the backup set. It interprets the expiration date as the date after which it can automatically write over the data on the volume. If the user mounts a volume for backup before its expiration date, and if that volume bears his account number, then BACKUP asks for confirmation before writing on the volume. If the volume does not bear the user's account number, BACKUP will not allow him to write on that volume. For magtapes, BACKUP also requests density (800 or 1600 bpi) and, for 7-track tapes, parity.

After answering the labeling questions, the user ensures that the volume is mounted. He write-enables the volume for a Backup operation. For a Restore operation, magtape volumes may be mounted write-locked; however, disk volumes must always be mounted write-enabled in order to allow updating of the bad block file. In response to the DEVICE? query, he types the device unit number on which the volume is mounted.

The following shows the mount procedure print-out:

```
PLEASE ENTER BACKUP SET NAME<BACK28>-
PLEASE ENTER EXPIRATION DATE<08-Jul-77>-
PLEASE ENTER DENSITY IN BPI<800>- 800
PLEASE ENTER THE PARITY<ODD>-
MOUNT    DEVICE:        MT    :
             ID:        BACK28
           SEQ#:        1
        DENSITY:        800 BPI
         PARITY:        ODD
        IDENTIFICATION WILL BE FINAL UPON SUCCESSFUL MOUNT
DEVICE? MM1:
THIS VOLUME HAS NO BACKUP LABEL!
MOUNT IT ANYWAY <NO>? Y
```

When BACKUP has finished using a volume, it prints a dismount message like the following:

```
DISMOUNT DEVICE:        MM1:
             ID:        BACK28
           SEQ#:        1
        DENSITY:        800 BPI
         PARITY:        ODD
        PLEASE LABEL THIS VOLUME!
```

When the dismount message appears, the user physically dismounts the volume (if necessary) and readies the next volume for processing. BACKUP requests name, expiration date, parity, and density for only the first volume in each backup set. For subsequent volumes, the user specifies only the device unit number.

## 18.8 BACKUP ERROR HANDLING

The BACKUP package may encounter four types of errors: dialogue command errors, interruption command errors, volume mount errors, and BACKUP processing errors.

Dialogue command errors occur during the Backup or Restore dialogue. The BACKUP package diagnoses these errors immediately. Usually, the user responds to such an error by typing the correct dialogue answer.

Interruption command errors can occur when the user types an interruption command. BACKUP responds immediately to these errors, too. The user simply types a valid command to correct the error situation.

Volume mount errors can occur when the BACKUP package mounts tape and disk backup volumes. Some volume mount errors are user errors (for example, specifying an illegal density setting for tape); others are related to the hardware (for example, tape errors that prevent labelling).

BACKUP processing errors can occur any time during the BACKUP run except during the dialogue. Processing errors can be related to the hardware on which the BACKUP is running or they can indicate logic errors.

The following four sections describe each type of error.

### 18.8.1 Dialogue Command Errors

When BACKUP encounters a dialogue command error (usually a syntax error), it prints an error message on the user's terminal. The error message is ?COMMAND ERROR followed by either a BACKUP-specific error message or a RSTS/E error message generated during the syntax processing of the command. After printing the error message, BACKUP prints a question mark and repeats its prompt. If the user responds with "? <CR>", BACKUP prints the command line up to the position of the error, then reprints the prompt. Table 18-4 describes the BACKUP dialogue error messages.

**Table 18-4 BACKUP Dialogue Error Messages**

| Message | Meaning |
|---|---|
| ?BAD DIRECTORY FOR DEVICE | The directory structure on the device the user specified is corrupt. The user should try to place the file on a different device. |
| ?CAN'T FIND FILE OR ACCOUNT | BACKUP cannot find the file or account the user specified. The user should ensure that he typed the filename and account correctly, and that they exist. |
| ?DEVICE HUNG OR WRITE-LOCKED | The device the user specified is write-locked or generated a read or write error. The user should write-enable the device (if necessary) or specify a different device. |
| ?DEVICE NOT AVAILABLE | The device the user specified is disabled or assigned to another user. The user should specify a different device. |
| ?DUPLICATE SWITCH | The file specification contains multiple ACCESS or CREATION comparisons. BACKUP accepts only one of each comparison in a file specification. |
| ?ILLEGAL EXCEPT NESTING | The file specification includes more than one EXCEPT comparison. BACKUP accepts only one EXCEPT in a file specification. |
| ?ILLEGAL FIELD | The response contains a field that is not permitted. For example, the user cannot specify a device name in the response to FROM FILES? |
| ?ILLEGAL FILENAME | The filename the user specified contains invalid characters or is in incorrect format. |
| ?ILLEGAL KEYWORD | The response contains a misspelled or incorrectly abbreviated keyword, or has incorrect punctuation marks. |

(continued on next page)

**Table 18-4 (Cont.) BACKUP Dialogue Error Messages**

| Message | Meaning |
|---|---|
| ?ILLEGAL OPERAND | The EXCEPT comparison in the file specification contains an unmatched parenthesis, or the day or year number in a date is out of range. |
| ?INCOMPLETE COMMAND FILE | The user typed CTRL/Z or the indirect command file ended before the expected end-of-file. BACKUP returns to the BACKUP OR RESTORE? query. |
| ?NO DEFAULT | The current dialogue question does not have a default answer. The user must type an explicit response. |
| ?NOT A VALID DEVICE | The device the user specified is not on this system. |
| ?NOT A VALID DEVICE (PROHIBITED) | The device the user specified is illegal in response to this question. |
| ?PROTECTION VIOLATION | The file or account the user specified is protected against him. |
| ?TOO MANY FILES OPEN ON UNIT | Only one file at a time can be open on a magtape unit. The user must specify the second file on another device. |
| ?TOO MUCH DATA | The response contains more data than BACKUP accepts for this question. |
| ?UNIT NUMBER NOT VALID | BACKUP does not accept a device unit number in the response to this question. |

### 18.8.2 Interruption Command Errors

An invalid interruption command may generate one of three error messages. Table 18-5 summarizes these messages and their meanings.

**Table 18-5 Interruption Command Error Messages**

| Text | Meaning |
|---|---|
| ?CAN'T DETACH | DETACH command is invalid because listing file is KB:. |
| ?UNRECOGNIZED COMMAND | The user typed a string that is not an interruption command. |
| ?ILLEGAL COMMAND | The user typed an interruption command that is illegal (for example, typing CONT when no PAUSE is in effect). The user can type LEGAL for a list of interruption commands that are currently legal. |

### 18.8.3 Volume Mount Errors

Several errors can occur during the volume mount process. BACKUP prints an error message, then repeats the current prompt. Table 18-6 summarizes these error messages.

**Table 18-6 BACKUP Volume Mount Error Messages**

| Message | Meaning |
|---|---|
| ?INVALID DATE | Expiration date must be in DD-MMM-YY format and cannot have already occurred. |
| ?INVALID DENSITY SETTING | Valid density settings are 800 and 1600 bpi. |
| ?INVALID PARITY SETTING | Valid parity settings are ODD and EVEN. |
| ?CAN'T WRITE ON THIS TAPE | BACKUP cannot write a label on the magtape because of tape errors. The user should try another magtape. |
| ?MAGTAPE SELECT ERROR | The magtape unit the user specified is not READY or is OFF-LINE. The user should ready the unit or specify another unit. |

### 18.8.4 BACKUP Processing Errors

BACKUP processing errors can occur as the package selects, transfers, compares and deletes files and generates the listing file. A processing error can be a hardware error, which indicates a hardware problem (for example, ?DEVICE HUNG OR WRITE-LOCKED), or a logic error, which indicates an attempt at an illogical or prohibited action (for example, ?PROTECTION VIOLATION). The BACKUP package contains error handling routines for common hardware and logic errors.

**18.8.4.1 Selection Errors** – If a logic error occurs during the selection process, BACKUP automatically skips over the file or account causing the error. BACKUP prints an error message, then proceeds to select the next file or account, according to the following rules:

1. If an error occurs while BACKUP is looking up a file, BACKUP skips that file and looks for the next sequential file.
2. If an error occurs during the search for an account, BACKUP terminates the search on that input volume and continues the search for that account on the next input volume.
3. If an error occurs during the search for an account on the final input volume, BACKUP returns to the first input volume and searches for the next sequential account.
4. If an error occurs in the final account on the final input volume, the selection process ends and BACKUP begins to transfer the selected files and accounts.

BACKUP contains error handling routines for the ?DEVICE HUNG OR WRITE-LOCKED hardware error. These routines allow the user to request that BACKUP retry the procedure that encountered the error. The user can sometimes recover from the error without missing any files or accounts because ?DEVICE HUNG OR WRITE-LOCKED errors occasionally indicate transient hardware problems. The user can also request that BACKUP skip over the error. If the user requests a skip over a ?DEVICE HUNG OR WRITE-LOCKED error, BACKUP ends the search for the current file or account and proceeds to look for the next file or account, as it does for a logic error.

If a bad block error (?USER DATA ERROR ON DEVICE) occurs during the selection phase of a Backup operation, Backup prints an error message, stops searching for the current file or account, and proceeds to look for the next file or account as for a logic error.

During a Restore operation, a bad block error at Index load time usually means that the index file is corrupt. Restore asks the user to mount a different volume (if necessary) so that the Restore operation can read from another index file. If an error occurs in the secondary index file, the BACKUP run is aborted. The user must then run BACKUP again and use the auxiliary index file. After errors occur in both the primary and secondary index files, BACKUP cannot recover from an error in the auxiliary index file.

A bad block error (?USER DATA ERROR ON DEVICE) that occurs in the Backup or Restore work-file is fatal. BACKUP prints ?UNEXPECTED ERROR – ?USER DATA ERROR ON DEVICE and the run is aborted.

**18.8.4.2 Transfer, Deletion, and Listing Errors** – Logic errors during the transfer, deletion, and listing processes are usually the result of failure to open a file that BACKUP must transfer or delete. For example, the logic error ?PROTECTION VIOLATION means that the protection code of a certain file prohibited its transfer or deletion by this user. Another logic error, ?SUPERSEDE FAILURE, indicates a failure to find or replace a file for which the user requested a supersede operation. The BACKUP package automatically skips over the file or account causing a logic error during transfer, deletion, or listing processes.

BACKUP error handling routines allow the user to retry or skip over a ?DEVICE HUNG OR WRITE-LOCKED error that occurs during the transfer, delete, or listing process. Retry and skip work in the same way here as they do during the selection process.

The ?SEQUENCING PROBLEM ON MAGTAPE error can occur during a Restore from magtape. This error means that Restore could not read the record numbers on the tape. When the sequencing error occurs, Restore usually prints several bad block error messages (as it tries and fails to read magtape records), then recovers automatically. The user cannot restore the file in which the sequencing error occurred, but he can restore the remainder of the files on the magtape.

If BACKUP encounters a bad block during a transfer operation that involves a RSTS/E disk, BACKUP prints an error message. It copies the rest of the current file although the file is corrupt. If the bad block is in the work-file, the BACKUP run is aborted.

The BACKUP package specially formats each backup disk when the user mounts it for a Backup operation. BACKUP creates a bad block file on the disk and allocates to this file all bad blocks it finds during formatting. The BACKUP package does not permit any disk that has an excessive number of bad blocks to be used as a backup disk. When a Backup or Restore operation discovers a bad block on a backup disk, it allocates the bad block to the bad block file. This procedure prevents future use of the block for backup data. (The system manager should be aware that BACKUP bad block files are not equivalent to the RSTS/E file BADB.SYS. The system manager must initialize a backup disk as a RSTS/E file structured disk before using it for Backup. The *System Manager's Guide* describes disk initialization procedures.)

**18.8.4.3 Informational Messages** – During the transfer process, BACKUP often prints informational messages. These messages usually do not indicate errors, but inform the user about files that are open (and subject to change) and files that have changed in length since selection. BACKUP transfers these files, but the copies may not be accurate. Other messages list files that BACKUP cannot transfer: those deleted since selection and those whose length is zero.

# CHAPTER 19
# FLOPPY DISK TRANSFER: THE FLINT PROGRAM

The system program FLINT (FLoppy disk INTerchange) allows the user to transfer information from IBM[1] floppy disks to a standard disk that RSTS/E can read (for example, DK:, DP:, SY:). Conversely, FLINT also allows the transfer of information from RSTS/E-readable disks to the floppy disks that an IBM system can read. Either of these two transfers may involve multiple floppy disks, referred to as volumes in the dialogue and in its description here. FLINT can transfer IBM floppy disk information to a RSTS/E disk only.

FLINT also allows the user to obtain a directory of an IBM floppy disk; this directory contains information related to the unique data format of an IBM floppy disk.

To perform these transfers and to print the directory of an IBM floppy disk, FLINT communicates with the user by dialogue. The program asks the user a series of questions which he answers one at a time. There are two different dialogues for the transfer operations, IBM-to-RSTS/E and RSTS/E-to-IBM. There is another dialogue, consisting of only two questions, for the printing of an IBM directory.

## 19.1 RUNNING FLINT: THE INITIATION COMMANDS
To run the FLINT program, the user types

```
RUN $FLINT
```

FLINT then responds by identifying itself and printing a number sign (#) to show its readiness to accept one of three initiation commands. Each of these initiates a different dialogue, as follows:

| **Command** | **initiates the dialogue for:** |
|---|---|
| /DIRECTORY | listing an IBM floppy disk directory |
| /TORSTS | transferring from an IBM floppy disk to a RSTS/E disk |
| /TOIBM | transferring from a RSTS/E disk to an IBM floppy disk |

Each command may be abbreviated, at the minimum, to its first three letters: /DIR, /TOR, /TOI. The user may also run FLINT by the CCL command FLI[NT] if the FLINT command was installed on his system at startup; see Section 19.7.

## 19.2 LISTING THE DIRECTORY OF AN IBM FLOPPY DISK
Typing /DIR in response to FLINT's number sign (#) prompt will cause FLINT to print a dialogue of two questions. In the following description of the dialogue, FLINT's questions are underlined and followed by explanations.

<u>OUTPUT TO?</u>

This question asks the user where he wishes FLINT to print the directory of the floppy disk(s). If he wishes it printed at his terminal, he may give a null response (i.e., press RETURN). If, however, he wishes to save the directory in a file, he must respond with an output file specification of the form

```
dev: [proj,prog] name.ext<prot>
```

[1]IBM is a trademark of International Business Machines Inc.

The user must specify at least the device and filename. By default, [proj,prog] is the current account and <prot> is <60>.

DIRECTORY OF?

This question asks the user to specify the floppy disk(s) for which the directory will be printed. A null response specifies the disk DX0:. Other disk specifications must be typed in the form

    DXn:

or

    n

where n = 0 to 7

Multiple floppy disk specifications must be separated by commas.

### 19.2.1 The Form of the Directory

This is a listing of an IBM floppy disk directory obtained by FLINT:

```
DIRECTORY OF DX1:

DSN      BRL      BOE      EOE      EOD      BI  MVI  VSN

DATA     080      01001    73026    01001

TOTAL OF 1 DATA SET ON DX1:
```

The abbreviated headings on the second line are listed and defined here, reading from left to right:

| | |
|---|---|
| 1. DSN | Data Set Name: the 1-8 character name a user has given the data set (corresponds to "filename" in RSTS/E) |
| 2. BRL | Block/Record Length: in each 128-position sector, the number of positions containing data |
| 3. BOE | Beginning Of Extent: the address of the first sector in the data set; output in the form TT0SS, where TT is track number and SS is sector number |
| 4. EOE | End Of Extent: the address of the last sector reserved for this data set (in the format as BOE above) |
| 5. EOD | End Of Data: the address of the next unused sector within the data set extent |
| 6. BI | Bypass Indicator: A blank in this field means the data set is intended for processing; a B means it is not. |
| 7. MVI | Multi-volume Indicator: A blank in this field means a data set is wholly contained on this floppy disk; a C means that a data set is Continued on another floppy disk; and L means that this floppy disk is the Last on which a continued data set resides. |
| 8. VSN | Volume Sequence Number: the sequence of volumes in a multi-volume data set. Blanks indicate that volume sequence checking is not to be performed. |

## 19.3 TRANSFERRING IBM FLOPPY DISK DATA TO RSTS/E

Typing /TOR in response to FLINT's number sign (#) prompt will cause FLINT to print the dialogue for transferring floppy disk data to a standard RSTS/E disk (DK0:, for example). In the following description of that dialogue, FLINT's questions are underlined and followed by explanations.

OUTPUT TO?

This question asks the user to specify the RSTS/E file that is to receive data from the floppy disk(s). The user must respond with an output file specification of the form

dev: [proj,prog] name.ext<prot>

By default, dev: is the system disk, [proj,prog] is the current account, and <prot> is <60>. The user is reminded that the device must be a disk (DP1:, for example).

TRANSLATE FROM EBCDIC TO ASCII <YES>?

This question asks the user if he wishes the floppy disk data to be translated from its current EBCDIC mode, unreadable to RSTS/E, into ASCII mode, readable to RSTS/E. As the default – enclosed in angle brackets – indicates, a null response (i.e., pressing RETURN) will cause the data to be transferred "in translation." If, however, the user types NO, FLINT will perform an "image mode" (byte-for-byte) transferral.

INPUT FROM?

This question asks the user to specify the floppy disk(s) from which data will be transferred to the output file specified in answer to the OUTPUT TO question. He may specify such an input floppy disk by typing a response of the form DXn: or simply n, where n is a device number from 0 to 7. Such a response will cause FLINT to transfer the first data set on the specified floppy disk. If however, the user gives a null response (i.e., presses RETURN), FLINT will transfer the first data set from DX0:.

If the user wishes FLINT to transfer a specific data set, he may give a response of the form DXn:DSN where DSN is the data set name, composed of one to eight ASCII characters, including blanks – spaces and tabs. Since these blanks are recognized as part of the IBM data set name, care should be taken in their use. If FLINT cannot find the specified data set, it will print the message FILE NOT FOUND and will repeat the INPUT FROM question.

If the user does not specify a data set name, FLINT will print a message of the form

dsn IS THE DATA SET BEING TRANSFERRED

where dsn is the name of the first data set on the first floppy disk specified.

FLINT examines the data set label – a header with statistical information – to determine if the data set resides on more than one floppy disk. If it does, and if the user has specified only one floppy disk, FLINT prints the message/question

dsn RESIDES ON MORE THAN ONE FLOPPY –
DO YOU WISH TO CONTINUE <NO>?

If the user responds by typing YES, FLINT will proceed to its next question. As the default <NO> indicates, however, a null response causes no transfer, and instead causes FLINT to print its # prompt.

FLINT next computes the blocking factor to be used in the transfer of the current data set, and prints a message expressing that factor as the ratio of IBM to RSTS/E records. This message has the form

```
N XXX CHARACTER IBM RECORDS = 1 RSTS RECORD
```

FLINT, after printing this message, proceeds with the transfer, extracting data from each specified input device.

(FLINT computes this blocking factor by determining how many of the 128 positions in each floppy disk sector contain data in the form of characters. This count is the IBM record size, and is used to divide the RSTS record size (512). The product of this division is printed as N in the above message. For example, if each IBM record contains 80 data characters, FLINT will find that 6 full IBM records can be placed in a 512-byte "RSTS record" [512/80 = 6.4]. Thus, "6 80 CHARACTER IBM RECORDS = 1 RSTS RECORD.")

**NOTE**

The first logical record on the RSTS/E output file is always reserved for a statistical header of the form described in Section 19.3.2. This header is 6 bytes long. Thus, if the logical record length of the dataset to be transferred is less than 6 bytes, FLINT will use as many logical records as are necessary to contain the header. If, for example, the data set's logical records are only 4 bytes long, FLINT will have to allocate 2 of those records to hold the header.

If the entire data set is not contained on the input device(s) specified, FLINT prints the question

```
NEXT INPUT DEVICE?
```

The user should respond with the number of the drive (m) on which the next portion of the data set resides. FLINT will continue to perform transfers and to print NEXT INPUT DEVICE requests until it determines that the current floppy disk contains the end of the specified data set.

Note that on multi-volume input FLINT continually checks data set names against the one specified by the user, and, if possible, also checks volume numbers for correct sequence. If, for example, data set IDAT is being transferred and the third input volume specified contains no IDAT, FLINT will print the message

```
FILE NOT FOUND – DXn:IDAT
```

and will repeat the message NEXT INPUT DEVICE.

Moreover, if the volumes contain sequence numbers, and the number on a particular floppy disk is not 1 greater than that on the previous floppy disk, FLINT will print the message

```
VOLUME #m CANNOT FOLLOW VOLUME #n
```

and will repeat the message NEXT INPUT DEVICE.

If no fatal errors occur, FLINT, on completing the IBM-to-RSTS/E transfer, will print the message

```
EXCHANGE COMPLETED
```

at the user's terminal, and will list both the number of IBM records read and the number of RSTS records written.

### 19.3.1 Specifying the Known Floppy Disks of a Data Set

Before he initiates the transfer of a multiple-volume data set, the user may know on which floppy disks the data set resides. In that case, he may respond to the INPUT FROM question by specifying the floppy disk on which the data set begins, then the data set's name, and then the other floppy disk(s) onto which the data set is continued. Here is an example of such a multiple-device response:

```
DX1:DATASETA,DX2:,DX3:,4
```

Note that this user, in accordance with good data procedure, has mounted the floppy disks containing DATASETA on consecutively numbered drives, and has specified them in order. A user may, on the other hand, mount and specify his floppy disks out of sequence, but risks confusion if he does. Note also that the unsequenced order of specifications must match exactly the unsequenced order of mounting. If it does not, FLINT will print the message VOLUME #[m] CANNOT FOLLOW VOLUME #[n], which is described earlier.

### 19.3.2 Format of the RSTS/E Disk

FLINT writes, on the RSTS/E output file, a header record. This header contains the following information (all fields are 2-byte integer fields in standard CVT%$ format; see the *BASIC-PLUS Language Manual*, Section 11.5):

| Byte | Contents |
|---|---|
| 1-2 | Physical block number of the last logical block in the file |
| 3-4 | Number of logical records in the last physical block |
| 5-6 | Logical record length in bytes |
| 7- | Remainder of the logical record contains no data (the entire logical record is reserved for the header) |

If the logical record length of the data set is less than 6 bytes, this header will occupy more than one RSTS/E logical record.

## 19.4 TRANSFERRING RSTS/E FILES TO IBM FLOPPY DISKS

Typing /TOI in response to FLINT's number sign (#) prompt causes FLINT to print the dialogue for transferring a RSTS/E file to IBM floppy disk(s). In the following description of that dialogue, FLINT's questions are underlined and followed by explanations.

OUTPUT TO?

This question asks the user to specify the IBM data set to which FLINT will output the RSTS/E data. The user must respond with a data set name in the form

```
DXn:DSN
```

DXn: specifies a floppy disk on which the data set is written; n is a device number from 0 to 7. DSN is the output data set name, composed of 1 to 8 characters, including blanks. If the user omits the DSN, FLINT assumes the name RSTS.

Multiple volumes must be separated by commas; for example:

```
DX0:DATSETC,DX2:
```

TRANSLATE FROM ASCII TO EBCDIC <YES>?

This question asks the user if he wishes the RSTS input data to be translated from its current ASCII mode into EBCDIC mode. As the default <YES> indicates, a null response will cause FLINT to transfer the data "in translation." If, however, the user types NO, FLINT will perform an "image mode" (byte-for-byte) transfer.

INPUT FROM?

This question asks the user to specify the RSTS/E file from which data will be transferred to the data set specified in answer to the OUTPUT TO question. The user must specify an input filename in RSTS/E format:

```
dev: [proj,prog] name.ext<prot>
```

By default, dev: is the system disk, [proj,prog] is the current account, and <prot> is <60>. Note that the device must be a RSTS/E disk.

RECORD LENGTH <128>?

This question asks the user to specify the number of characters to be included in each IBM output record. As the default indicates, a null response will cause the output records to contain 128 characters each. (IBM records contain 1 to 128 characters and are fixed length.) To specify a shorter record length, the user may respond by typing any number smaller than 128 and greater than 0.

On receiving this response, FLINT computes the blocking factor to be used in the transfer of RSTS/E data, and prints a message expressing that factor as the ratio of IBM to RSTS/E records. This message has the form:

```
1 RSTS RECORD = N IBM RECORDS
```

For an explanation of how FLINT computes the blocking factor, see "INPUT FROM?" in the IBM-to-RSTS/E dialogue (Section 19.3).

After computing the blocking factor, FLINT proceeds with the transfer. If FLINT determines that the RSTS/E file cannot fit on the number of floppy disks specified in answer to the OUTPUT TO question, it will print the question

VOLUME #(N + 1)?

This question asks the user to specify an additional floppy disk for output; he may respond with DXn: or with n, where n in either case is a device number from 0 to 7. Or, he may give a null response, which is equivalent to DX0:. Until FLINT can complete the transfer, it will continue to ask the VOLUME # question and to accept one of these responses. All floppy disks used in the transfer will be labelled with volume sequence numbers, and, where necessary, with continuation markers.

NOTE

No more than 99 floppy disks can be used for one data set. If the user attempts to use more, FLINT will print the message

```
MAXIMUM NUMBER OF VOLUMES EXCEEDED
```

and will issue its # prompt.

If no fatal errors occur, FLINT, on completing the RSTS/E-to-IBM transfer, will print at the user's terminal a message of the form

```
EXCHANGE COMPLETE
data set name RESIDES ON x FLOPPIES
```

where x is the number of floppy disks on which the named data set resides.

## 19.5 DIALOGUE EXAMPLES: /DIRECTORY, /TORSTS, AND /TOIBM

In the first example, the user requests the directory of a floppy disk, then transfers data from that disk to a RSTS/E file. The user's input is underlined.

```
FLINT   V06B-03   RSTS   V06B-02   TIMESHARING
#/DIREC
OUTPUT TO?
DIRECTORY OF? DX0:

DIRECTORY OF DX0:

DSN            BRL    BOE       EOE       EOD       BI  MVI  VSN

XFILEXMA       128    01001     30011     30012
WUMPUSDB       080    30012     43019     43020
PHONE DB       128    43020     47003     47004

TOTAL OF 3 DATA SETS ON DX0:


#/TORS
OUTPUT TO? PHONE.DAT
TRANSLATE FROM EBCDIC TO ASCII <YES>?
INPUT FROM? DX0:PHONE DB

DX0:
   4  128 CHARACTER IBM RECORDS = 1 RSTS RECORD

EXCHANGE COMPLETED
  88  IBM RECORDS READ
  89  LOGICAL ( 23 PHYSICAL ) RSTS RECORDS WRITTEN
```

In the /DIRECTORY dialogue, the directory of DX0: is printed at the user's terminal.

In the /TORSTS dialogue, data set PHONE DB is transferred from floppy disk DX0: to the RSTS/E output file PHONE.DAT; by default, the data is translated into ASCII on being transferred. FLINT computes and prints the blocking factor (4 IBM records to 1 RSTS/E record).

In the next example, the user transfers a RSTS/E file to an IBM floppy disk. The user's input is underlined.

```
FLINT   V06B-03   RSTS   V06B-02   TIMESHARING
#/TOIBM
OUTPUT TO? DX1:MYDATA
TRANSLATE FROM ASCII TO EBCDIC <YES>?
INPUT FROM? MYDATA.DAT
RECORD LENGTH <128>?

1 RSTS RECORD = 4 IBM RECORDS

EXCHANGE COMPLETED
MYDATA RESIDES ON 1 FLOPPY
```

In this /TOIBM dialogue, the RSTS/E file MYDATA.DAT is transferred to the IBM data set MYDATA on floppy disk DX1:. By default, the data is translated to EBCDIC on being transferred. FLINT computes and prints the blocking factor (1 RSTS/E record to 4 IBM records).

## 19.6 FLINT ERROR MESSAGES

FLINT, on encountering an error, prints the appropriate error message. If the error is non-fatal, FLINT repeats the prompting question; if the error is fatal, FLINT aborts the transfer. Table 19-1 lists and describes the error messages specific to FLINT. Error messages specific to RSTS/E are in Appendix C.

**Table 19-1 FLINT Error Messages**

| Message | Meaning |
|---|---|
| ?DATA SET NAME TOO LONG | The IBM data set name specified has more than the allowed maximum of eight characters, including spaces and tabs. |
| ?DATA SET NOT FOUND | FLINT cannot find the specified data set on the current floppy disk. |
| ?ILLEGAL DATA SET NAME | A data set name has been specified where none is permissible. |
| ?ILLEGAL EXTENSION | The extension specified contains unacceptable characters or violates the specification format. |
| ?ILLEGAL FILE NAME | The filename specified contains unacceptable characters or violates the specification format. |
| ?ILLEGAL PPN | The specified account number contains wildcards, which are not allowed. |
| ?LOGICAL DEVICE NAME NOT ASSIGNED | The logical device name specified has not been previously assigned. |
| ?OUTPUT DEVICE MUST BE DISK | In a /TORSTS transfer, an output device other than the only permissible type – a RSTS/E disk – has been specified. |
| ?VOLUME #[m] CANNOT FOLLOW VOLUME #[n] | In a multi-volume /TORSTS transfer, the specified input volume number [m] is not 1 greater than the previously specified volume number [n]. |
| The following errors are fatal and cause the transfer to be aborted: | |
| ?BOE = EOD – TRANSFER ABORTED | In the data set specified, the Beginning Of Extent address is the same as the End Of Data address; i.e., the data set has no length. |
| ?DATA SET LABEL MUST BE HDR1 – TRANSFER ABORTED | The label ID of a data set specified for transfer is not HDR1, which it must be according to IBM format. |
| ?TRACK 00 DOES NOT CONTAIN ERMAP FIELD – TRANSFER ABORTED | There is no ERMAP field in track 00 of the specified floppy disk; according to IBM format, this field must be present. |

## 19.7 FLINT AS A CCL COMMAND

The user may run FLINT (if the CCL command has been installed) by first typing the CCL command FLI[NT], then typing one of the dialogue initiation commands – /DIR[ECTORY], /TOR[STS], or /TOI[BM] – depending on which dialogue he wishes to run. Note that all these commands, including the CCL, may be minimally abbreviated to their first three letters. Here are three examples of CCL command strings:

```
FLI/DIRECT

FLIN/TOR

FLINT/TOIB
```

/DIRECT initiates the dialogue for listing a floppy disk directory, /TOR the dialogue for IBM-to-RSTS/E transfer, and /TOIBM the dialogue for RSTS/E-to-IBM transfer.

# CHAPTER 20
# DEVICE CONTROL PROGRAMS

## 20.1 SETTING TERMINAL CHARACTERISTICS: THE TTYSET PROGRAM

The TTYSET system program is used to establish terminal characteristics for the user terminal. TTYSET can be run by any user before or after he is logged into the system. If the terminal is not logged into the system, only one command can be entered to the TTYSET program at a time in the following format:

```
    SET xxxx
```

where xxxx represents one of the TTYSET commands shown in Table 20-1 and is entered with the RETURN or ESCAPE key. If the user is logged into the system, he can run TTYSET as follows:

```
    RUN $TTYSET
    TTYSET V06B-03 RSTS V06B-02 TIMESHARING
    TERMINAL CHARACTERISTICS PROGRAM
    ? XXXX
```

where xxxx is again one of the TTYSET commands shown in Table 20-1. When xxxx has been entered to the system with the RETURN or ESCAPE key, TTYSET again prints ? to accept additional commands. To return control to BASIC-PLUS command level, the user types EXIT, CTRL/C, or CTRL/Z.

**Table 20-1 RSTS/E TTYSET Commands**

| Command | Function |
|---|---|
| WIDTH n | Set the width of the print line for this terminal to n which can be between 1 and 254. As a result, the system automatically generates a CR and LF sequence if n printing characters have been printed and another printing character is to be transmitted. |
| TAB | Enable hardware tab control. System transmits HT characters without translation. |
| NO TAB | Disable hardware tab control. To move to the next tab stop, the system transmits the correct number of space characters instead of transmitting an HT character. |
| FORM | Enable hardware form feed and vertical tab control. System transmits FF and VT characters without translation. |
| NO FORM | Disable hardware form feed and vertical tab control. System transmits 4 line feed characters in place of a FF or VT character. |
| LC OUTPUT | System transmits the lower case characters CHR$(96) through CHR$(126) inclusive to the terminal without modification. |
| NO LC OUTPUT | System translates any lower case character to its upper case equivalent before transmitting it to the terminal. |
| XON | Special terminal hardware allows the computer to interrupt transmission of characters from the terminal by sending the terminal an XOFF character CHR$(19). Similarly, the computer instructs the terminal to resume transmission of characters by sending the terminal an XON character CHR$(17). The terminal hardware must |

(continued on next page)

**Table 20-1 (Cont.) RSTS/E TTYSET Commands**

| Command | Function |
|---|---|
| XON (Cont.) | respond to XOFF and XON characters by ceasing and resuming transmission, respectively. |
| NO XON | Terminal does not have hardware required for XON feature. |
| LOCAL ECHO | Terminal, or its acoustic coupler, echo prints characters as they are generated locally. System does not echo characters received from such a terminal. |
| FULL DUPLEX | Characters generated are sent only to the computer. System therefore echoes each character received so that it will be printed locally and translates certain characters to perform the proper action. For example, a CR character received is echoed as a CR and LF character sequence. |
| SCOPE | Terminal uses a CRT display and the following characteristics:<br>1. Conforms to synchronization as described under the STALL command,<br>2. System echoes a DEL character (RUBOUT) as backspace, space, and backspace sequence.<br>3. System generates NUL characters as fill for timing the following operations: home, erase to end of screen, erase to end of line, direct cursor addressing, and line feed. |
| NO SCOPE | Terminal is not a CRT display device. The system echoes a DEL character (RUBOUT) by printing a \ character and the last character typed and removes the last character typed from the terminal input buffer. Subsequent DEL characters cause the next to last characters to be sequentially printed and removed from the terminal input buffer until a character other than DEL is received. As a result, the system echoes another \ character to delimit the erased characters and echoes the correct character. |
| LC INPUT | Terminal transmits the full ASCII character set and system does the following:<br>1. Recognizes only the ESC character CHR$(27) as an escape character,<br>2. Echoes and uses the CHR$(125) and CHR$(126) characters without translation, and<br>3. Echoes and uses lower case alphabetic characters without translation. |
| NO LC INPUT | System treats ESC, }, and ~ characters (CHR$(27), CHR$(125), and CHR$(126) respectively) as escape characters and translates lower case characters received to their upper case equivalents. |
| NO FILL | System does not generate fill characters for any characters transmitted. |
| FILL LA30S | Set the fill characteristics for a serial DECwriter (LA30S). |
| FILL n | Set the fill factor to n for this terminal where n is between 0 and 6. As a result, the system generates a multiple of fill characters for each hardware control character it transmits. See Section 20.1.3 for a discussion of generalized fill characters. |
| SPEED n | Set to n the rate at which the terminal's interface can accept or pass characters. The value for n can be any number defined for the keyboard number in the system file TTYSET.SPD. This command can be used only on lines whose interfaces can be programmed to handle more than one speed (e.g., DH11, DZ11). |

(continued on next page)

**Table 20-1 (Cont.) RSTS/E TTYSET Commands**

| Command | Function |
|---|---|
| SPLIT SPEED i/o | Set to i the rate at which the terminal's interface passes input to the computer and set to o the rate at which the terminal's interface accepts output from the computer. The values i and o must be defined for the keyboard number in the system library file TTYSET.SPD. This command can be used only on lines whose interfaces can be programmed to handle more than one speed (e.g., DH11, DZ11). |
| NO PARITY | System ignores the parity bit on characters it receives and treats the parity bit on characters it transmits to the terminal as if it were a data bit. |
| EVEN PARITY | System sends characters to the terminal with the parity bit properly set for even parity but ignores the parity bit on characters received. |
| ODD PARITY | System sends characters to the terminal with the parity bit properly set for odd parity but ignores the parity bit on characters received. |
| STALL | Terminal obeys the following synchronization standard: if the terminal sends an XOFF character (equivalent to the CTRL/S combination), the computer interrupts transmission until the terminal sends either an XON character (equivalent to the CTRL/Q combination) or any other character. If the system receives an XON character, it does not keep that XON character as data. If the system receives another character, it resumes transmission and keeps that character as data. No characters are lost. |
| NO STALL | XON and XOFF characters sent by the terminal have no special meaning. |
| UP ARROW | System echoes a control and graphic character combination as the ^ or ↑ character (CHR$(94)) followed by the proper graphic. For example, CTRL/E prints a ^ E or ↑E. |
| NO UP ARROW | System echoes control and graphic character combination as is. |
| PRINT filename | Sends the specified file to the terminal, in binary mode. This command's special use is in initializing a terminal for which special software settings must be made: setting 8 spaces to a TAB, for example. In such a case, 8 escape sequences would be placed in the file to be specified with PRINT. |
| ESC SEQ | System treats an ESC character CHR$(27) as an indication of the start of an incoming escape sequence. The character is not echoed, nor are the characters in the sequence. The processing of input escape sequences is described in the *RSTS/E Programming Manual*, section 4.4.2. |
| NO ESC SEQ | System treats an ESC character CHR$(27) as a line terminating character and echoes it as a $ character. |
| DELIMITER x | Makes the printable character x a private delimiter for use with GET statements. Character will not be echoed. |
| DELIMITER "x" | Makes the nonprintable character x (TAB, for example) a private delimiter for use with GET statements. This format also works for printable characters. Character will not be echoed. |
| DELIMITER | Disables the private delimiter. Private delimiter is also disabled by any macro (multiple characteristic) command. |

(continued on next page)

**Table 20-1 (Cont.) Device Control Programs**

| Command | Function |
|---|---|
| ESC | System treats only ASCII 027 (decimal) code as ESCAPE (ESCAPE or ALTMODE key). |
| NO ESC | System treats ASCII 027, 125, 126 (decimal) as ESCAPE (ESCAPE or ALTMODE key). |
| EXIT | Terminate a program and return control to the monitor. |
| HELP | Print a listing of single and macro commands. |

In addition to the commands described in Table 20-1, TTYSET allows the user to set all the individual characteristics for a certain type device by executing one command called a macro command. Each macro command assigns predefined characteristics, any of which can be altered in turn by proper use of an individual characteristic command. Table 20-2 describes the default values for each macro command.

The TTYSET program checks commands before and during execution and reports errors by printing the messages described in Table 20-3. If any errors involve the terminal speed file, the user should immediately notify the system manager or responsible system programmer.

### 20.1.1 ESCAPE, ALTMODE, and PREFIX Characters

The RSTS/E system translates certain ASCII characters in a special manner. Some terminals have the outmoded ASCII character keys ALTMODE (125 decimal) and PREFIX (126 decimal). More recently designed terminals incorporate the 1968 ASCII character set and include the following control characters:

ESCAPE = 27 (decimal)
} = 125 (decimal)
~ = 126 (decimal)

The RSTS/E system interprets CHR$(27) as a line terminating character and automatically translates any CHR$(125) and CHR$(126) characters input from a terminal into 27 (decimal). The system thus treats 125 (decimal) and 126 (decimal) as line terminators.

A user having a terminal with the 1968 ASCII character set can make the system treat 125 (decimal) and 126 (decimal) characters as they are printed rather than as control characters. The TTYSET commands LC INPUT and NO LC INPUT;ESC alter internal parameters for a given terminal so that the system treats 125 (decimal) and 126 (decimal) as they are printed or translates them automatically to their upper-case equivalents.

### 20.1.2 Lower and Upper Case Characters

Some terminals can send and print both lower case and upper case characters. Such terminals can therefore print the echo returned by the system to give the user an accurate visual representation of the character transmitted.

Terminals such as the VT05 alphanumeric display can send either lower case or upper case characters but can print only upper case characters. Consequently, the echo response of such a terminal to a lower case character is the upper case counterpart. The terminal gives the user no visual indication that the character transmitted was a lower case character.

Terminals such as the ASR-33 and KSR-33 type devices neither send nor receive lower case characters. If such a terminal receives a lower case character, it prints the corresponding upper case character.

Normally, RSTS/E software translates all lower case characters to their upper case counterparts before processing them. To take advantage of different features of terminal hardware, a RSTS/E user chooses or omits lower case translation depending upon whether the terminal produces lower case output, lower case input, or both. The LC OUTPUT and LC INPUT commands, respectively, omit translation of lower case characters generated as output on and input from a terminal. NO LC OUTPUT causes lower to upper case translation on output to the terminal; NO LC INPUT causes lower to upper case translation on input from the terminal.

**Table 20-2 Default Single Characteristic Settings**

| Individual Characteristic | Macro Command | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **ASR33** | **KSR33** | **ASR35** | **KSR35** | **VT05** | **VT05B** | **LA30** | **LA30S** | **2741** | **VT50** | **VT50H** | **VT52** | **VT55** | **RT02** | **LA36** |
| WIDTH n | 72 | 72 | 72 | 72 | 72 | 72 | 80 | 80 | 130 | 80 | 80 | 80 | 80 | 32 | 132 |
| TAB/NO TAB | NO TAB | NO TAB | TAB | TAB | TAB | TAB | NO TAB | NO TAB | TAB | TAB | TAB | TAB | TAB | NO TAB | NO TAB |
| FORM/NO FORM | NO FORM | NO FORM | FORM | FORM | NO FORM | NO FORM | NO FORM | NO FORM | NO FORM | NO FORM | NO FORM | NO FORM | NO FORM | NO FORM | NO FORM |
| LC OUTPUT/ NO LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT | LC OUTPUT |
| XON/NO XON | XON | NO XON | XON | NO XON | NO XON | NO XON | NO XON | NO XON | NO XON | NO XON | NO XON | NO XON | NO XON | NO XON | NO XON |
| LOCAL ECHO FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX | FULL DUPLEX |
| SCOPE/ NO SCOPE | NO SCOPE | NO SCOPE | NO SCOPE | NO SCOPE | SCOPE | SCOPE | NO SCOPE | NO SCOPE | NO SCOPE | SCOPE | SCOPE | SCOPE | SCOPE | NO SCOPE | NO SCOPE |
| LC INPUT/ NO LC INPUT | NO LC INPUT | NO LC INPUT | NO LC INPUT | NO LC INPUT | NO LC INPUT | NO LC INPUT | NO LC INPUT | NO LC INPUT | LC INPUT | NO LC INPUT | NO LC INPUT | LC INPUT | LC INPUT | NO LC INPUT | LC INPUT |
| FILL n FILL LA30S | 0 | 0 | 1 | 1 | 0 | 3 | 0 | FILL LA30S | 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| SPEED n SPLIT SPEED i/o 2741 | 110 | 110 | 110 | 110 | 300 | 150/2400 | 300 | 300 | 2741 | 300/1200 | 300/1200 | 300/1200 | 300/1200 | 110 | 300 |
| NO PARITY EVEN PARITY ODD PARITY | NO PARITY | NO PARITY | NO PARITY | NO PARITY | NO PARITY | NO PARITY | NO PARITY | NO PARITY | ODD PARITY | NO PARITY | NO PARITY | NO PARITY | NO PARITY | EVEN PARITY | NO PARITY |
| STALL/ NO STALL | STALL | STALL | STALL | STALL | STALL | STALL | STALL | STALL | NO STALL | STALL | STALL | STALL | STALL | NO STALL | STALL |
| UP ARROW NO UP ARROW | UP ARROW | UP ARROW | UP ARROW | UP ARROW | UP ARROW | UP ARROW | UP ARROW | UP ARROW | NO UP ARROW | UP ARROW | UP ARROW | UP ARROW | UP ARROW | NO UP ARROW | UP ARROW |
| ESC SEQ/ NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ | NO ESC SEQ |
| ESC/NO ESC | NO ESC | NO ESC | NO ESC | NO ESC | ESC | ESC | ESC | ESC | NO ESC | ESC | ESC | ESC | ESC | ESC | ESC |
| DELIMITER 0 | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER | DELIMITER |

**NOTE**

None of the macro commands in Table 20-2 performs an automatic PRINT.

**Table 20-3 TTYSET Error Messages**

| Message | Meaning |
|---|---|
| ?<string> IS AN ILLEGAL KB SPECIFICATION | The keyboard number denoted by <string> is not between 0 and 63 or is not a valid number. |
| ?ILLEGAL FILL FACTOR | Fill factor specified is not between 0 and 6. |
| ?ILLEGAL WIDTH | Width specified is not between 1 and 254. |
| ?COMMAND <s ring> ILLEGAL | Command indicated by <string> is undefined. |
| ?ILLEGAL SPEED | Speed given is not one defined for the device in the $TTYSET.SPD file. |
| %WARNING – ERROR READING $TTYSET.SPD FILE | Program warns user of possible corruption in the terminal speed file and denotes the exact error by printing the COMMAND ERROR message. |
| ?COMMAND ERROR – <text> | The RSTS error denoted by <text> was encountered when executing the command. |
| ?ERROR – <text> | The RSTS error denoted by <text> was encountered when executing the system function to change terminal status. |
| %WARNING – CANNOT OPEN $TTYSET.SPD FILE | Program could not access the terminal speed file and denotes the exact error by printing the COMMAND ERROR message. |

## 20.1.3 Generalized Fill Characters

The RSTS/E system automatically generates a variable number of NUL characters (CHR$(0%)) as fill characters after outputting certain control characters. The generation of these meaningless NUL characters allows the terminal sufficient time to complete the physical action initiated by the control character, thus permitting the terminal to synchronize itself properly for printing the next meaningful character. The values in Table 20-4 interrelate the numbers of NUL characters generated for control characters at certain TTYSET characteristic settings.

**Table 20-4 Generalized Fill Characters**

| Control Character | Decimal Value | Scope | No Scope | Form | No Form | Tab | No Tab |
|---|---|---|---|---|---|---|---|
| CR | 13 | 0 | 2741: POS/10 + 1; NOT 2741: $1 \times 2^{\text{Fill-1}}$; Fill LA30s* | N/A | N/A | N/A | N/A |
| LF | 10 | $1 \times 2^{\text{Fill-1}}$ | 0 | N/A | N/A | N/A | N/A |
| HT(tab) | 9 | N/A | N/A | N/A | N/A | $1 \times 2^{\text{Fill-1}}$ | Spaces are sent. |
| VT(Vert. Tab) | 11 | $1 \times 2^{\text{Fill-1}}$ | See FORM/NO FORM | $5 \times 2^{\text{Fill-1}}$ | Do 4 LFs | N/A | N/A |
| FF | 12 | N/A | N/A | $9 \times 2^{\text{Fill-1}}$ | Do 4 LFs | N/A | N/A |
| CTRL/N (direct cursor addressing VT05 | 14 | $1 \times 2^{\text{Fill-1}}$ | 0 | N/A | N/A | N/A | N/A |

* Function of type head's position at time of CR; this function is non-linear.

**NOTE**

If the fill factor is 0, no fill characters are ever generated. The expressions in Table 20-4 do not apply for FILL 0.

The TTYSET command FILL n sets the fill factor to be used in the above table; the command NO FILL or FILL 0 disables the automatic generation of fill characters.

In Table 20-4, the fill factor is used as an exponent of 2. When the fill factor is increased by 1, therefore, the number of fill characters generated is doubled. Most terminals require no fill characters at low baud rate settings, But at some baud rate (for example, 600 baud for VT05s), a terminal starts to require fill characters. As the baud rate increases, the number of fill characters required also increases.

Since the common baud rates increase by doubling, one would increase the fill factor by 1 each time the baud rate doubled. (For example, the appropriate fill factor for a VT05 at baud 600 is 1; at 1200, 2; at 2400, 3; etc.)

### 20.1.4 XON/XOFF Remote Reader Control

To operate a low speed paper tape reader connected to the RSTS/E system by either a data set (dial up) or a communications line, two requirements must be fulfilled as follows.

1. The terminal must be equipped with the requisite hardware option for XON/XOFF remote reader control.
2. The XON/XOFF feature must be enabled for the given terminal.

The user can selectively enable and disable remote reader control for a remote terminal by the TTYSET commands XON and NO XON.

For low speed readers connected to the system by a local line interface, none of these requirements is necessary.

### 20.1.5 Output Parity Bit

The RSTS/E software always ignores the parity bit on characters received from a terminal and omits the parity bit on characters it transmits. Since some terminals not supplied by DEC can receive even or odd parity characters, the software must be conditioned to send parity characters. The TTYSET commands EVEN PARITY, ODD PARITY, or NO PARITY condition the software to send the correct parity. The software ignores parity bits on characters input to the system.

### 20.1.6 Private Delimiters

TTYSET enables the user to establish, on his keyboard, a special delimiter for use in BASIC-PLUS GET statements. Creating such a private delimiter is especially useful on a data entry terminal with a specialized keyboard: the user can designate a large or conveniently located key as the delimiter key. The user should note, however, that a character designated as a private delimiter will not echo at the terminal. This rule applies to nonprintable as well as printable characters; TAB, for instance, will have no visible effect.

To make a private delimiter of a printable character, the user types the SET DELIMITER command followed by the character, either with or without enclosing quotation marks. The following examples illustrate the format:

```
SET DELIMITER A
```

or

```
SET DELIMITER "A"
```

Either of these commands makes the character A into a private delimiter.

To make a private delimiter of a nonprintable character, the user types the SET DELIMITER command followed by the nonprintable character enclosed in quotation marks. In the following example, the user makes the TAB character into a private delimiter:

```
SET DELIMITER "          "
```

To disable the private delimiter, the user simply types the SET DELIMITER command:

```
SET DELIMITER
```

The private delimiter is also disabled by any macro (multiple characteristic) command: for example, SET VT05, SET LA36, etc.

**20.1.6.1 Limitations of Private Delimiters** – As previously mentioned, a character designated as a private delimiter will not echo at the terminal. Thus, making a private delimiter of a common formatting character such as TAB may cause confusion in editing a program.

The SET DELIMITER command causes the existing ASCII code for the specified character to be overridden. Therefore, if the character normally has expanded or alternate functionality, that functionality will be eliminated by the command. RUBOUT, for example, will neither backspace (on a display terminal) nor print backslashes (on a hardcopy terminal).

Private delimiters work only in the GET statement – not in the INPUT or INPUT LINE statement.

### 20.1.7 SET as a CCL Command

Under a standard RSTS/E system, the user can execute TTYSET by the correct concise command language (CCL) command. To execute a CCL command, the user issues the proper CCL command followed by the desired TTYSET command(s). Multiple TTYSET commands on the same line are separated by semicolons (;). For example,

```
SET LA36;WIDTH 80
```

For more information on CCL commands, see the introductory material in Chapter 13.

## 20.2 MOUNTING AND DISMOUNTING PRIVATE DISKS: THE UMOUNT PROGRAM

A non-privileged or privileged user can execute the MOUNT or DISMOUNT commands of the UMOUNT system program to logically mount and dismount private disk packs and cartridges. The commands are executed only if the standard CCL feature is available on the system. Otherwise, the system prints the ?WHAT? error message. An attempt to run the UMOUNT program by any other means generates the message PLEASE USE THE 'MOUNT' OR 'DISMOUNT' COMMAND.

A non-privileged or privileged user executes the MOUNT or DISMOUNT CCL command to logically mount or dismount private disk packs and magtapes. To logically mount a disk pack, the user types the MOUNT command in the following format:

```
MOUNT dev:pack id label/option(s)
```

This command logically associates the disk pack on the specified drive with the specified identification label, and applies the specified option(s).

To logically dismount a disk pack the user types the DISMOUNT command in the following format:

```
DISMOUNT dev:pack id label
```

This command logically dismounts the disk pack on the specified drive, and removes the specified pack identification label from the system's table of logical names. The following sections explain both procedures in detail.

### 20.2.1 Logically Mounting a Disk Pack or Cartridge

The following steps are taken to logically mount a private pack or cartridge on the system:

1. The user loads the pack or cartridge in the available drive, and ensures that the device is available by running the SYSTAT program.
2. The user then ensures that the drive is write enabled.
3. The user then types the MOUNT command followed by the device designator, the appropriate identification label, and any option(s) desired.

The following example illustrates the format of the MOUNT command:

```
MOUNT DK1:MYPACK/LOGICAL:STAT

READY
```

As a result of this command, the UMOUNT program runs and logically associates RK drive unit 1 with the identification label MYPACK, and, as the option specifies, with the logical name STAT. (Disk options are described in the following section.) The READY message indicates that the disk is available for read and write access and in the UNLOCK state.

An attempt to write a file on the disk when the current account is nonexistent generates the ?DISK PACK IS PRIVATE error. The user must ask the system manager or responsible system programmer to create the current account on the disk.

If any error occurs in the device designator or in the identification label, the program prints the following error message:

```
ERROR IN MOUNT – <text>
```

The notation <text> represents the related RSTS/E error encountered. On printing this message, the program terminates.

If any error occurs in mounting the disk on the system, the program prints the following error message:

```
?PROCESS ERROR IN MOUNT – <text>
```

The notation <text> represents the related RSTS/E error encountered. Typical errors include having the device protected against writing (?DEVICE HUNG OR WRITE LOCKED), specifying an incorrect identification label (?PACK IDS DON'T MATCH), and trying to mount a disk which is already mounted (?DISK IS ALREADY MOUNTED.)

If the program encounters an error when attempting to unlock the disk to enable write access, it prints the following message:

```
    ?PROCESS ERROR IN UNLOCK
```

The notation <text> represents the related RSTS/E error encountered. The pack, however, is mounted and available for read access only. The user should report any error of this kind to the system manager or responsible system programmer.

### 20.2.2 MOUNT Options: Disk Pack or Cartridge

The options described in this section may be used with the MOUNT command when mounting a private disk. Options are summarized in Table 20-5.

If the disk to be mounted must be kept ~~locked,~~ the user appends the /LOCK option to the MOUNT command, as in the following example:

```
    MOUNT DB1:DATA1/LOCK

    READY
```

As a result of this command, UMOUNT runs and logically associates the identification label DATA1 with RB unit 1. The READY message indicates that the disk pack is in the LOCK state. Only privileged users have read and write access to the disk pack.

If the disk to be mounted is to be read only, the user appends the /READ ONLY option to the MOUNT command, as in the following example:

```
    MOUNT DK1:MYPACK/RONLY
```

As a result of this command, the disk may be referred to as DK1: and MYPACK, and can be read but not written.

If a logical name other than the pack identification label is desired for the disk, the user appends the option /LOGICAL:, followed by a logical device name, to the MOUNT command. The following example illustrates:

```
    MOUNT DK1:MYPACK/LOGICAL:PACK3
```

As a result of this command, the disk may be referred to either as DK1: or PACK3.

If only the physical device specification is to be allowed for the mounted disk, the user appends the option /NOLOGICAL to the MOUNT command, as in the following example:

```
    MOUNT DK1:MYPACK/NOLOGICAL
```

As a result of this command, the disk may be referred to only by its physical specification, DK1:. No logical names are allowed.

If a public disk is to be mounted as private, the user appends the option /PRIVATE to the MOUNT command, as follows:

```
    MOUNT DK1:PUB001/PRIVATE
```

As a result of this command, the disk is mounted as a private disk, and is not used as part of the public structure. The disk may be referenced as PUB001 and as DK1:.

### 20.2.3 Assigning a Magtape Unit, and Using Options

The MOUNT command can be used with a magtape designator to assign a unit to the current job and to set default labeling format. The options described in this section are summarized in Table 20-5.

To set the magtape labeling format, the user appends the option /DOS or /ANSI to the MOUNT command. The following example shows the procedure:

```
MOUNT MT1:/DOS

READY
```

As a result of this command, magtape unit 1 is assigned to the current job with DOS/BATCH labeling default. The /ANSI option sets the labeling default to ANSI standard format.

If the current job is privileged, the /JOB:n option can be used with the MOUNT command to reassign the unit to job number n. The following example shows the procedure:

```
MOUNT MT1:/JOB:5

READY
```

As a result of this command, magtape unit 1 is reserved to job 5 with the currently assigned default labeling format. If job 5 is not active, an error message is printed. If an unassigned logical name is given for the magtape unit in the MOUNT command, UMOUNT prints the error text ?ERROR IN MOUNT – ?ILLEGAL DEVICE and returns to BASIC-PLUS command level. If a name is given with the magtape designator in the MOUNT command, UMOUNT prints the message ?ERROR IN MOUNT – ?SYNTAX ERROR and returns to BASIC-PLUS command level.

### 20.2.4 Logically Dismounting Disk Packs, Cartridges, and Magtapes

The following steps are taken to logically dismount a private pack or cartridge:

1. The user determines the number of OPEN files on the device by running SYSTAT. If non-zero, he waits until all files are closed before proceeding. If zero, he proceeds.
2. The user then types the DISMOUNT command followed by the device designator of the drive and the pack identification label.

The following example illustrates the format of the DISMOUNT command:

```
DISMOUNT DK1:PACKID

READY
```

As a result of this command, the UMOUNT program runs and logically dismounts the pack on RK drive 1; the pack identification is removed from the system's table of logical names. The READY message indicates that the drive is free for other usage and that the user can safely remove the pack or cartridge from the drive.

If the program encounters an error when it attempts the dismount action, it prints a message in the following format:

```
?PROCESS ERROR IN DISMOUNT – <text>
```

The notation <text> represents the related RSTS/E error encountered. A typical error is attempting to dismount a disk which has open files (?ACCOUNT OR DEVICE IN USE).

To rewind a magtape and take it off line, the user appends the /UNLOAD option to the DISMOUNT command, as in the following example:

```
DISMOUNT MT0:/UNLOAD
```

As a result of this command, the magtape on unit 0 is rewound and placed off line.

**Table 20-5 The UMOUNT Options**

| Device | Option | Function |
|---|---|---|
| disk pack | /LOCK | keeps the mounted disk locked. |
| disk pack | /LOGICAL | allows a logical name other than the pack ID for the mounted disk. |
| disk pack | /NOLOGICAL | disallows logical name references to the mounted disk; allows only physical specification. |
| disk pack | /PRIVATE | allows a public disk to be mounted as private. |
| disk pack | /RONLY | allows the mounted disk to be read only; prevents all write access. |
| magtape | /ANSI | sets magtape labeling default to ANSI format. |
| magtape | /DOS | sets magtape labeling default to DOS format. |
| magtape | /JOB:n | reassigns magtape unit to job number n (current job must be privileged). |
| magtape | /UNLOAD | (with DISMOUNT command) rewinds a magtape and takes it off line. |

# CHAPTER 21
# USING SYSTEM SPOOLING SERVICES: THE QUE PROGRAM

The QUE system program creates requests, or jobs, which are to be executed by spooling programs. QUE also executes auxiliary operations such as listing pending requests, killing pending requests, and modifying pending requests. The QUE program checks the syntax and validity of each request and passes it to another program for further processing if the request is valid. If any part of a request is invalid, the program rejects the entire request, prints an appropriate error message, and allows the user to try again.

For the system to execute a request created by QUE, the system must contain three other programs: a queue manager program (QUEMAN), an operator services program (OPSER), and a spooling program. QUEMAN processes a request created by QUE and places it in the system queue file QUEUE.SYS. OPSER communicates between the operator and the spooling programs. The particular spooling program executes requests on its related device when the QUEMAN program passes it a pending request. Spooling programs on standard RSTS/E systems can be SPOOL, BATCH, or RJ2780. If no requests are pending in the queue file, the spooling program executes a sleep operation until activated by QUEMAN.

The file QUEUE.SYS is stored under the system library account on the public structure. The file can accommodate a total of 250 job requests.

## 21.1 RUNNING QUE AT A TERMINAL

A privileged or non-privileged user runs QUE at a terminal logged into the RSTS/E system by typing the RUN command as follows:

```
RUN $QUE
QUE   V06B-03   RSTS V06B-02   TIMESHARING
```

When QUE runs, it prints a header containing the program name, its version and revision numbers, and the system name and version and revision numbers. The program ensures that the system queue file exists, and is accessible on the system library account. If no errors occur, QUE prints a pound sign (#) prompt to signal the user that he can type a command. QUE prints this pound sign after processing each command.

If QUE runs and encounters an error when accessing the queue file QUEUE.SYS, it prints the following message and terminates.

```
?QUEUE NOT INITIALIZED
READY
```

The READY prompt is printed by the system after QUE has terminated. To run QUE without an error, the user must have the system manager run the QUEMAN program and initialize the queue file.

The user may terminate QUE by typing E or the CTRL/Z combination in response to the pound sign character, as shown below:

```
#E
READY
```

Control is returned to the BASIC-PLUS command level, as indicated by the READY prompt. Commands to queue, list, kill, and modify jobs and the related error messages are explained in the following sections. A summary of the commands appears in Table 21-1.

**Table 21-1 QUE Program Commands**

| Command | Format | Description |
|---|---|---|
| Q | Q device:jobname=file spec(s) | Allows user to give file specification(s) of a file to be processed. A comma is used to separate file specifications if more than one is given. See Table 21-2 for options which may be given with a file specification. |
| L | L device:=jobname(s) | Prints at the terminal a list of the currently pending requests for the spooling program. If device: is not given, LP0: is assumed. If device: is not given and jobname(s) is given, the = character is required. Valid devices are LP:, LP0:-LP7:, BA:, BA0:-BA7:, and RJ:. Specifying LP: or BA: indicates all units of that device. If no jobname appears, QUE prints all jobs. |
| K | K device:=jobname(s) | Removes from the queue the job(s) indicated by device specification and jobname. If device: is not given, QUE assumes LP0:. If device: is not given and jobname(s) is given, the = character is required. |
| M | M device:jobname/options | Modifies the parameters of a job already in the queue. The following parameters can be modified by the M command: priority, forms name, AFTER date or time, type of forms control, number of job copies, job status, and MODE. |
| H | H | Causes an information message to be printed at the terminal. |
| E | E | Causes QUE to terminate and to return control to BASIC-PLUS command level. |
| CTRL/Z | The user types the CTRL/Z combination | Causes QUE to terminate (same as E). |

## 21.2 USING THE QUE COMMAND

The Q command typed in response to the pound sign creates a request for a spooling program. The Q command has the following general format.

```
Q device:jobname=file spec,file spec,. . . ,file spec
```

where:

| | |
|---|---|
| device: | is the device designator LP:, LP0:-LP7:, BA:, BA0:-BA7:, and RJ: and unit number of the device which the spooling program services. If a designator is not given, LP0: is assumed. If a device is specified, the = sign is required. The designators LP: and BA: cause the request to be queued for any available spooling program for that device type. |

If a unit number is specified, the request is queued for that unit and is executed only if a spooling program is running on that unit. Additionally, the general batch processor BA: processes only those jobs queued for BA: and not jobs queued for BA0: through BA7:. BATCH processors BA0: through BA7: however, process requests queued for their respective units and requests queued for BA:.

| | |
|---|---|
| jobname | is a maximum of six alphanumeric characters to identify a user's request in the queue. If device: and jobname are not specified, the = sign is optional and the filename of the first file specified in the request is the job name. Several options described in Table 21-2 may accompany the jobname. |
| file spec | is the external file specification of the file to be processed from the queue and any combination of Q command options. Up to 11 files can be included in a request. File specifications are separated by a comma. (See Table 21-3 for the options.) |

The jobname (or the left hand side of the = character) can have several output options which apply to the entire request. Each option must be preceded by a slash character (/). Options may be minimally abbreviated to their first two letters; for example, /MO, /AF, /PR, etc. Table 21-2 lists and describes the output options.

**NOTE**

The filename options /CO:nn, /NH, /DE, and /BI may also be specified as job options. (These are described in Table 21-3.) Any filename option so specified is applied to each file in the job.

The external file specification of a file to be included in the job can have the following form:

```
device:filename.extension [proj,prog] /option/option . . .
```

A valid device is either a valid disk designator or null (which indicates the system device in the public structure). QUE interprets logical names if the user makes the assignment before QUE runs. Otherwise, an unassigned logical device name generates the ILLEGAL INPUT FILE error. See Section 5.4 for a description of logical names.

The filename and extension may consist of ? and * characters described in Section 12.3.1 of this guide. A file specified with no extension is given the default extension .LST if it is in a line printer queue, or the default extension .CTL if it is in a BATCH queue. The [proj,prog] designation is the project-programmer number (account) under which the file to be printed is stored. The designation can be omitted if the file is stored under the current user's account. The /option designation is a slash character (/) followed by one of the Q command options. The options are described in Table 21-3.

The following command typed to QUE,

```
#Q ABC.DAT
#
```

causes the file ABC.DAT stored under the current user's account on the public structure to be sent to the queue manager for printing on the spooled line printer, unit 0. SPOOL generates one copy of the file under the user job header ABC. If printing of the file is interrupted for any reason, the program continues printing after the last character printed before the interruption.

**Table 21-2 QUE Job Output Options**

| Option Format | Description |
|---|---|
| /MODE:n | The value n specifies the MODE which the spooling program will use in its OPEN statement for the output device. The following specific MODE options may be used instead of /MODE:n (see the *RSTS/E Pocket Guide*). |
| /LENGTH:nnn | The value nnn (1 to 127) sets form length in lines per page. |
| /CNVERT | Change character 0 to character O. |
| /TRUNCATE | Truncate lines longer than unit's configured length. |
| /LPFORM | Enable software formatting. |
| /UPPERCASE | Translate lower- to upper-case characters. |
| /SKIP | Skip 6 lines at bottom of each form. |
| /AFTER:dd-mmm-yy:hh:mm | The value dd-mmm-yy specifies the date, month, (abbreviated to 3 letters, as in JAN), and, optionally, the year (77 for example) after which the queue manager will initiate the current request. For yy, the default is the current year. The value hh:mm specifies, in hours and minutes, the 24-hour (military) time after which the queue manager will initiate the current request. For example, /AFTER:10-JAN-77:13:30 causes the job to be initiated after 1:30 p.m. on January 10, 1977.<br><br>If the time is specified without the date, the current date is assumed. |
| /PRIORITY:n | QUE sets the priority to the value n which can be between 0 and 255. If /PRIORITY:n is not specified, QUE assigns the value 128. A privileged user can specify a priority between 0 and 255. A non-privileged user can specify a priority between 0 and 127. The priority setting determines the job's place in the queue; the higher the number, the higher the priority. If two or more jobs have the same priority, they go to the spooler in order of job requests. |
| /TYPE:xxx | Use the format specified by xxx for printing the file. Value may be:<br><br>FTN FORTRAN forms control<br>CBL COBOL forms control<br>EMB Embedded forms control (equivalent to paper tape)<br>IMP Implied forms control (LF and CR printed before each record) |
| /FORMS:<form name> | The string <form name> specifies the name of the form on which the job is printed; <form name> is up to 6 characters, of which the first character should not be a digit. The default form name is NORMAL. |
| /JCOPIES:n | The value n is the number of job copies to be printed. If /JCOPIES:n is not specified, one copy of the job is printed. |

**Table 21-3 Q Command Options**

| Option | Format | Description |
|---|---|---|
| Copies | /CO:nn | Tells SPOOL to print the number of copies indicated by the decimal integer n. If /CO:n is not given, one copy is printed. |
| Noheader | /NH | Tells SPOOL to suppress printing of the file header. |
| Delete | /DE | Tells SPOOL to delete the file after it is printed if the protection code of the file permits. If /DE is not specified, the file is left intact. The QUEMAN program checks the protection code of the file. |
| More | /MORE | Used only at the end of a command line to indicate to QUE that the user has more text to type on the next physical line. |
| Binary | /BI | Used to indicate a binary file to the RJ2780 program, which must be in TRANSPARENT mode. |

To queue a file from a device other than the public disk, the user specifies the device designator in the command. For example,

```
#Q DK1:A,BAS/CO:2
#
```

The indicated file on DK1: is queued for printing on line printer unit 0. SPOOL generates two copies of the file under the job header A.

To specify a job name for the request, the user types the name and the = character in the command. For example,

```
#Q SORT=DK1:FILEA,FILEB/CO:2
#
```

As a result, SPOOL prints SORT in the job header and generates one copy of DK1:FILEA.LST and two copies of DK1:FILEB.LST.

To specify a spooling program other than the one for line printer unit 0, the user types the appropriate designator and the = character in the command. For example:

```
#Q LP1:=DK1:FILOUT.001
#
```

As a result, QUE creates a request for FILOUT.001 on line printer unit 1, with job name FILOUT.

To specify a request longer than one physical line, the user types the /MORE option as the last item on the line. For example,

```
#Q LP1:PRINT1=FILE1.001,ABCDEF.001,/MORE
MORE > HELP.TXT,ACCT.SYS
#
```

As a result, QUE prints, as a prompting indicator, the text MORE>, after which the user may continue typing the request. QUE allows up to eleven files in one request. The user may type /MORE as many times as necessary. Note that /MORE is invisible to the parser; the completed command is interpreted as if it were typed as one line. Therefore, all punctuation (commas, colons, etc.) must appear in the appropriate places.

When a device designator is specified in the command, QUE uses that device (and not the system device) as a default for subsequent input file names on the physical line. For example, when this command is executed:

```
#Q LP1:SORT=DK2:FILEA,FILEB
```

QUE sets the device to DK2: for both FILEA.LST and FILEB.LST. Because no device designator is specified for FILEB, DK2: becomes the default device.

To set a priority to a job when queuing a file, the user specifies the /PR:n option with a value between 0 and 255. Without the /PR option, QUE assigns a priority of 128 to the job. The queue management program processes jobs with a priority of 128 or greater before processing jobs less than 128. Only privileged users can specify a priority greater than 128. With such a priority scheme, the queue manager can process important jobs before routine jobs and can delay processing less important jobs until routine jobs are completed.

## 21.3 USING THE L COMMAND

The L command lists, at the user's terminal, information about pending requests for a specified device. The following example illustrates the command and its results:

```
#L LP0:

LP0 QUEUE LISTING          28-Oct-76        04:12 PM
UNIT     JOB               S / P      FILES
 0       NOTICE[2,201]/SE:1038/FO:NORMAL
                           SK/128/TY:EMB
                                      SY :[1,2]NOTICE.TXT

 0       BNFTRN[2,201]/SE:1039/FO:NORMAL
                           0 /128/TY:EMB
                                      SY :[2,201]BNFTRN.SIF
                                      SY :[2,201]BNFBLD.SIF
                                      SY :[2,201]PERCNT.BAS

 0       VISITS[2,201]/SE:1040/FO:NORMAL
                           0 /128/TY:EMB
                                      SY :[2,201]VISITS.LST


#
```

**NOTE**

Because both QUE and QUEMAN can access a queue list simultaneously, QUE tests the linkage of the queue list to verify that the queue list has not been changed by QUEMAN during QUE's search. If QUE detects that the linkage has changed, QUE will display the following message and will restart the listing from the beginning of the queue list.

```
***** QUEUE CHANGING – WILL RESTART LISTING *****
```

The L command causes QUE to print two header lines for the specified spooled device. The first header line contains the device and unit designator of the specified device (LP0: by default), the current system date, and the time of day.

The second header line contains headings that denote the unit queued to, the job name and account number of the requester, and the status, priority, and full file specifications associated with each job. If no jobs are queued, QUE prints the header lines followed by the # character.

Jobs are listed in the order in which QUEMAN accesses them. In the first line of each job description, the jobname is followed by its account number and the sequence number (/SE:n) assigned the job by QUEMAN. Following the sequence number is the /FO [RMS] option and its argument, the forms name: the default NORMAL or the name specified by the user.

In the second line of each job description, the status appears under the heading S, and is indicated by one of the following:

| | |
|---|---|
| 0 | Job is in queue waiting to be processed. |
| S | Job has been sent to spooler. |
| A | Job is waiting for an /AFTER date/time to expire. |
| H | Job is in hold status: it has been modified (by the M command) to be put into hold, or it was put into hold if upon restarting QUEMAN found that the job was previously sent to the spooler. |
| K | Job is in kill status. |

Some status indicators may appear together; for example, SK means that the job was sent to the spooler, and that a K command was later issued for it.

Next on the second line appears the priority (0 to 255), under the heading P; 128 is the default. Following the priority are any job options specified by the user. Appearing under the lines describing a job are the descriptions of individual files in that job – the full file specifications followed by any file options specified by the user.

To list jobs on all line printers, the user types the L LP: command. QUE prints the related unit number in the UNIT column for each queued job. Jobs queued for the BATCH processor are listed by typing the L BA: command; jobs queued for the RJ2780 program are listed by typing the L RJ: command.

To list only information about one request or several requests, the user includes the = character followed by the jobname(s) in the L command. The following example illustrates:

```
#L LP0:=REW[200,57]
LP0 QUEUE LISTING          09-Nov-76        03:22 PM
UNIT     JOB               S / P        FILES
 0       REW    [200,57]/SE:2775/FO:NORMAL
                           S /128/TY:EMB
                                        SY :[200,57]EXPENS.BAS
                                        SY :[200,57]VISITS.LST/C:3
```

QUE prints the header lines and the data for the jobname specified. If the user specifies more than one jobname, he must separate them by commas, as in the following example:

```
#L LP0:=REW[200,57],REF[200,57]
LP0 QUEUE LISTING          09-Nov-76        03:22 PM
UNIT     JOB               S / P        FILES
 0       REW    [200,57]/SE:2775/FO:NORMAL
                           A /128/TY:EMB
                                        SY :[200,57]EXPENS.BAS
                                        SY :[200,57]VISITS.LST/C:3

 0       REF    [200,57]/SE:2776/FO:NORMAL
                           S /128/TY:EMB
                                        SY :[200,57]CHOICE.BAS/C:3/N
                                        SY :[200,57]CURZ  .LST/C:7/D

#
```

## 21.4 USING THE K COMMAND

The K command removes a job or jobs from the queue file QUEUE.SYS for a specified device. To remove a job, the user specifies the device designator and the = character followed by the job name or job names separated by commas. For example,

```
#K LP1:=PRINT1,BATCH1
#
```

As a result, each job for the user's account in the LP1: part of the queue with the name PRINT1 or BATCH1 is deleted if it is not currently being processed by the LP1: spooling program. In the case of matching jobnames, the user may specify the job to be removed from the queue by including the /SE: nnnnn option in the K command. The argument nnnnn represents the job's sequence number, assigned to it by QUEMAN (see Section 21.3).

## 21.5 USING THE M COMMAND

The M command allows a user to modify the parameters of a job that is already in the queue. By issuing M with appropriate options, the user can modify the job parameters originally set in the Q command: priority, MODE, date/time, number of job copies, FORM, and TYPE. If, however, the job has already been sent to a spooling program, QUEMAN will not allow any modifications.

To specify modifications to the job, the user types the device designator followed by the jobname and the desired modification option(s) in the following format:

```
M device:jobname/option(s)
```

Each option must be preceded by a slash (/). The following list contains job modification options that may be specified with the M command. Except where otherwise indicated (for example, with /HOLD and /UNHOLD), these options correspond in name, function, and format with those of the Q command. Their full descriptions are in Table 21-2.

| | |
|---|---|
| /HOLD | Halt the job's advancement in the queue, and do not send it to a spooling program until the /UNHOLD option is specified. |
| /UNHOLD | Continue the job's advancement in the queue. |
| /SE:nnnnn | Modify only the job with sequence number nnnnn (assigned by QUEMAN; see Section 21.3). Used in case of matching jobnames. |
| /PRIORITY:nnn | (See Table 21-2 for explanations of this and the remaining options listed here.) |

```
/MODE:nnnn or
     /LENGTH:nnn          /UPPERCASE
     /CONVERT             /SKIP
     /TRUNCATE
     /LPFORM

/AFTER:dd-mmm-yy:hh:mm

/TYPE:xxx

/FORMS:form name

/JCOPIES:n
```

## 21.6 CHAINING TO QUE FROM A USER PROGRAM

To run the QUE program by a CHAIN operation, the user program must first put in core common a string in the following format.

| | |
|---|---|
| Program name | The filename specification of the program to which QUE passes control. The specification can include a project-programmer field but not an extension. This program need not be the calling program. |
| Delimiter | The delimiter must be CHR$(13), the CR character. |
| Line number | An integer number in CVT%$ format indicating the line number of the program to which QUE passes control. For example, CVT%$ (28000%). |
| Command | The string QUE executes. It must have the same syntax as if it were typed at the terminal in response to the # character but must not include CR and LF characters. The option /MORE must not be used. |
| Delimiter | The delimiter must be CHR$(13). |
| Text | Optional string which QUE places in core common when it completes processing and passes control. Its length is restricted by the number of bytes remaining in core common. |

Secondly the user program must execute a CHAIN "$QUE" 31000 statement, to actually pass control to QUE stored in the system library account.

Upon completing the request passed to it through core common, QUE places a string in core common and executes a CHAIN "program name" line number statement. The program name and line number are taken from the values passed to QUE in core common. The core common string QUE passes has the following format.

| | |
|---|---|
| Error number | The QUE error code in CHR$ format. For example, CHR$(E%). See Section 21.6 for a description of the error codes generated by QUE. CHR$ (0%) indicates no error code was generated. |
| Error message text CHR$(13%) | If an error occurred, this string will be returned. |
| Text | The optional string which the user program passes to QUE. |

## 21.7 ERROR MESSAGES AND CODES

QUE reports an error condition by printing a message or by returning an error code. When running at a terminal, QUE prints a unique message which describes the error condition. No part of the command typed is executed. The user must retype the entire command in the correct format. Any RSTS/E system errors encountered are reported in the ?ILLEGAL INPUT FILE message. When running by means of a CHAIN operation from a user program, QUE reports an error by an error code when it passes control. Table 21-4 gives both the messages and codes and describes the related error conditions.

## 21.8 RUNNING QUE BY CCL COMMANDS

A user can run QUE by means of the concise command language (CCL) command QUEUE, which may be minimally abbreviated to QU. The correct format is QUEUE/<command line> where <command line> is any valid QUE program command. If the QUE program does not find a / character immediately following the characters QUEUE, it defaults the command to a Q command. For example, the command

```
QUE/Q JOB1 = ABC.DAT
```

**Table 21-4 QUE Error Messages and Codes**

| Error Code | Message | Meaning |
|---|---|---|
| 0 | None | QUE processed command without error. |
| 1 | ?INVALID COMMAND – <text> | The <text> indicated is an undefined command. |
| 2 | ?SW ON WRONG SIDE OF COMMAND | The option switch specified in the command line to QUE is reserved for the input side and was found on the output side, or vice versa. |
| 3 | ?INVALID SWITCH FORMAT | The option contains an invalid request. |
| 4 | ?UNDEFINED SWITCH | The command line contains an undefined option switch. |
| 5 | ?TOO MANY FILES | No more than eleven files can be given in the Q or K command. |
| 6 | ?NULL FILE SPEC | At least one file specification must be given in a Q or K command and none was found; or two commas occurred with no file specification between them. |
| 7 | | Not used. |
| 8 | ?ILLEGAL INPUT FILE – <text – filespec> | The request is invalid because the file indicated by the file specification caused the RSTS error described in the text. The user must retype request. |
| 9 | ?WILDCARDS NOT ALLOWED IN PPN | An asterisk is not allowed in the project-programmer field. |
| 10 | ?QUEUE FULL | The queue file is temporarily full. The user should try again when the spooling program has processed some pending requests. |
| 11 | ?BAD SWITCH VALUE – <text> | There is an invalid or improper number in the option indicated by <text>. For example, user specifies /MODE:A or a non-privileged user specifies /PR:129 or greater. |
| 12 | ?'MORE' REQUESTED ON A CHAIN | The /MORE option is not allowed when a program runs QUE by a CHAIN operation. |
| 13 | ?NOT A QUEUABLE DEVICE | The user attempted to queue a request to a device which cannot handle queued requests. For example, Q LG:=JOB. |
| 14 | | Not used. |
| 15 | ?QUEUE NOT INITIALIZED | The queue file QUEUE.SYS does not exist in the system library account. A privileged user must run QUEMAN to initialize the queue. |
| 16 | ?QUEMAN NOT RUNNING – CAN'T QUE OR KILL | The user has attempted to queue or kill a job and, because the queue manager is not running on the system, QUE generates an error. The user can, however, list jobs when QUEMAN is not running. |
| 17 | ?QUEUEING DISABLED | QUEMAN is in the shutdown process. |
| 18 | ?ILLEGAL SWITCHES FOR CMD | Specified options are illegal for commands used; e.g., /HOLD and /UNHOLD are illegal for Q. |

has the same effect as the command

```
QUE JOB1 = ABC.DAT
```

These commands are equivalent since both enter JOB1 in the queue for line printer 0. In either case, the QUE program runs and executes the command. If no error is detected, the system prints the READY message. An error in the command causes QUE to print the related error messages on a subsequent line, after which the system prints READY.

To include an output option when queuing a job, the user must begin the QUE command with /Q or else must give an explicit job name in the QUE command. For example,

```
QUE/Q/MODE:2048=DK1:DISK.BAS
READY
```

The / character preceding the Q character differentiates the Q command from the job name Q.

To list or kill jobs, the user must type QUE, followed by a slash (/) character and the appropriate command. For example,

```
QUE/L DISPLY[1,253]
LPO QUEUE LISTING         09-Nov-76        03:22 PM
UNIT     JOB              S / P       FILES
 0       DISPLY[1,253]/SE:558/FO:NORMAL
                          0 /128/TY:EMB
                                      SY :[1,253]DISPLY.V6B

Ready
```

QUE runs and executes the command indicated following the / character, after which control is returned to BASIC-PLUS command level.

## 21.9 RUNNING QUE AT A LOGGED-OUT TERMINAL

The currently pending requests for a spooling program can be printed by typing the QUE/L dev: command. For example, to print the line printer unit 1 queue, the user types the following command.

```
QUE/L LP1:
```

QUE executes the L (LIST) command for the LP1: queue. After printing the pending requests, QUE exits to RSTS/E and prints BYE.

# CHAPTER 22
# THE BATCH PROCESSING PROGRAM: BATCH

The ability to execute a stream of commands allows the user to submit jobs to be run without terminal dialogue. Batch processing is particularly useful in data processing operations that do not require interaction.

Batch input can be submitted from standard files on a random access device. For purposes of this description, input is dealt with as though it were on cards, where each card represents one record. Such input consists of elements of the batch control language and is collectively referred to as a batch stream. It is possible to execute multiple streams simultaneously by running multiple copies of the BATCH program. The capability to run more than a single batch stream is controlled by the system manager.

Sections 22.1 through 22.3 discuss the elements of the batch control language. Operating procedures are described in Section 22.4.

## 22.1 CONTROL STATEMENTS

Batch control language statements consist of a command field, specification field(s) and a comment field, in the following format:

```
$<command-field> [specification-field(s)]  [!comment]
```

Fields must be separated by one or more spaces and/or tabs.

A command field must always be present and may contain switch modifiers to control or limit the command. When appropriate, the command field is followed by one or more specification fields. The ! character is a comment prefix signifying that the information between the ! character and the line terminating character is a comment. The system takes no action on comment information.

The hyphen (-) character can indicate a continuation of a command. If a hyphen is the last character in a command, the next line is treated as a continuation of the previous line and must begin with a $ followed by a blank. The hyphen must appear before any comment.

A physical command line can have a maximum of 120 characters.

Double quote characters may be used in control statements to reproduce some text identically and override any special interpretation of characters by BATCH. For example, the exclamation point (!) in RSTS/E is the designator for the auxiliary library account [1,3] or for an installation defined account. In BATCH, the exclamation point signifies a comment. To prevent BATCH from misinterpreting the ! character given as an account designator, the user should include quotation marks as shown in the following sample control statement.

```
$RUN "!UPDAT"
```

As a result, BATCH executes the program UPDAT from the auxiliary library account. Without the quotes in the preceding example, the current program is executed and the characters following the ! character are treated as comment characters.

### 22.1.1 Command Field

The command field consists of the following elements:

1. A $ (dollar sign) character is in the first character position. The $ character is the control statement recognition character.
2. The command name begins in the second character position and immediately follows the $ character. For example, $JOB. No blanks are allowed in the command field because a blank (or horizontal tab) delimits the command field.
3. Valid switches follow the command name. No blank can appear between the command name and a switch. Switches are denoted by a slash (/). For example, /NAME=COMPL.

**NOTE**

Command names and switch names can be shortened to their first three characters. For example, the system interprets the command BAS as well as BASIC. Any other form, such as BASI, is invalid.

Multiple, adjacent blank characters are equivalent to one blank character. A horizontal tab is equivalent to one blank character. A blank character delimits a field; otherwise the blank character is ignored.

### 22.1.2 Specification Fields

A list of specification fields immediately follows the command field delimiter. The following rules apply:

1. Specification fields are separated by blanks.
2. Specification fields are terminated by a ! character if followed by a comment, or are otherwise terminated by a line terminating character.
3. Depending on the command, a specification field consists of a device specification, a file specification, or an arbitrary ASCII string, any of which can be followed by appropriate switches. The / character signals the start of a switch. For example, XYZ.BAS/SOURCE. The switch indicates that the file is a source file.

### 22.1.3 Comments

The following rules govern comment fields.

1. The start of a comment is defined by an ! character in the control statement.
2. Any character following an ! character and preceding the end-of-line terminator is treated as a comment and is otherwise ignored by the batch processor. Comment lines with no text may force line spacing on the job log and thereby make the log more readable. To force line spacing, the user includes lines consisting solely of $! followed immediately by a carriage return/line feed.

### 22.1.4 Syntactical Rules

The following are syntax rules for control language statements.

1. A control statement must have a command name (except in the case of the comment line "$!"). If the command name is omitted, the command is ignored. An unrecognizable command name is illegal, and causes the batch processor to print an error message. Only two forms of the command are recognized: the full name and the 3-character abbreviation of the name. For example, these are the legal BASIC-PLUS commands:

   ```
   $BASIC          $BAS
   ```

   Switches in the command field apply to the entire command. If a switch in the specification field contradicts a command field switch, an error results.

2. An asterisk is allowed in the filename or the file type field of a file specification, subject to restrictions on individual commands. See Section 22.2 for the description of file specifications. An asterisk can refer only to files already created. An asterisk appearing in a specification of a file not yet created constitutes an invalid file specification.

   The batch processor uses the leftmost 6 characters from file name fields longer than 6 characters and uses the leftmost 3 characters from the file type fields longer than 3 characters.

3. Switches can be used in the command field and specification fields of a control statement. Switches appearing in the command field are command qualifiers, and their function applies to the entire command. Switches appearing in specification fields apply only to the field in which they appear.

   Unrecognizable switches invalidate the control statements in which they appear.

### 22.1.5 Syntax Example

The following are sample control statements which illustrate the syntax of Batch statements.

```
$JOB/NAME=SMYTHE !FIRST JOB
$!
$!COMPILATION OF NEW SOURCE FILES
$!
$MESSAGE STARTING COMPILATIONS
$BASIC XYZ/SOURCE XYZ.LIS/LIST XYZ/EXECUTE
$BASIC ABC/SOURCE ABC.LIS/LIST ABC/EXECUTE
$!
$MESSAGE STARTING LISTING OUTPUT
$!
$PRINT *.LIS! ALL LIST FILES
$!
$EOJ
```

## 22.2 FILE SPECIFICATIONS

A file specification appears in the specification field and is an alphanumeric string containing the following elements:

filename.type

The batch processor assigns default values if part or all of the file specification is optionally omitted.

### 22.2.1 Filename Specification

A filename specification is a string of alphanumeric characters of which the first six characters must be unique. An asterisk in place of a filename denotes all files of the specified type in the account designated. If necessary, the batch processor generates a default filename related to the time of day as described in Section 22.2.3.

### 22.2.2 File Type Specification

A file type specification consists of a period, immediately followed by a string containing three or more alphanumeric characters, the first three of which must be unique.

The file type reflects the nature of the file. For example, a BASIC source file has .BAS as its file type. An asterisk in place of a file type denotes all file types including files with no type specified.

Some standard file types are listed below.

| | |
|---|---|
| .CTL | Batch control file |
| .DAT | Data file |
| .DIR | Directory file |

**Table 22-1 BATCH Special Characters**

| Character | Meaning |
|---|---|
| space | Separates fields in a control statement. Otherwise ignored unless embedded in string delimited by quotation marks. |
| horizontal tab | Separates fields in a control statement. (Equivalent to one space (blank) character; otherwise ignored.) |
| hyphen (-) | As last nonblank character in a control statement, indicates a continuation line follows. If the statement contains a comment, the hyphen must be last nonblank character before exclamation point. |
| exclamation point (!) | Indicates a comment unless embedded in a string delimited by quotation marks. |
| dollar sign ($) | Used as first character in first position of a control statement; causes control statement recognition. |
| slash (/) | Denotes a switch (separates specification field from switch name). |
| asterisk (*) | Indicates wild card in filename or file type. |
| colon (:)<br>equals (=) | Separate switch name from argument. |
| quotation marks (")<br>single quotation mark (') | Used to open and close a string to preserve embedded spaces or to pass a special character (such as !) without interpretation by BATCH. |
| plus (+) | Indicates file concatenation in $COPY statement. |
| comma (,) | Separates file, device, and/or account specifications within a specification field which allows multiple elements. |

| | |
|---|---|
| .BAS | BASIC-PLUS source file |
| .LIS | List file |
| .BAC | BASIC-PLUS compiled output file |
| .CBL | COBOL source file |
| .OBJ | COBOL or FORTRAN compiled output file |
| .SRT | PDP-11 SORT11 input, output or listing file |
| .MAP | Map file |
| .TMP | Temporary file |
| .B2S | BASIC-PLUS II source file |
| .TSK | BASIC-PLUS II executable file |
| | BASIC-PLUS II task built executable file |
| .FOR | FORTRAN source file |
| .SAV | FORTRAN linked executable file |

These file types are the defaults when no file type is specified. The default chosen is determined by the current operation and by the type of file expected. Table 22-2 summarizes the default file types that apply to particular batch commands.

Table 22-2 Batch Commands – Related Default File Types

| Command/ Section | Default Type | Meaning |
|---|---|---|
| $BASIC 22.3.3 | .BAS | Input source file default type. |
| | .BAC | Output executable file default type. |
| | .LIS | Listing file default type. |
| $CREATE 22.3.4.5 | .DAT | The file generated as output by CREATE has a file type of .DAT. |
| $DIRECTORY 22.3.4.4 | .DIR | The file in which the directory is to be recorded has a file type of .DIR. |
| $FORTRAN | .FOR | Input source file default type. |
| | .OBJ | Output object file default type. |
| | .LST | Listing file default type. |
| | .SAV | Executable linked file. |
| $JOB 22.3.1 | .CTL | Batch control file default type; assumed when the Batch job is on a file-structured device. |
| | .LOG | Batch output log file default type. |
| $PRINT 22.3.4.3 | .LIS | Default type of file to be printed. |
| $RUN 22.3.5 | .BAC | Default type of file to be run. |
| $COBOL 22.3.11 | .CBL | Input source file default type. |
| | .OBJ | Output object file default type. |
| | .LIS | Listing file default type. |
| $SORT 22.3.12 | .SRT | Input, output or listing files default type. |

### 22.2.3 File Specification Defaults

Defaults are assigned to omitted filename and file type elements as shown in Table 22-3.

### 22.2.4 Switch Specification

Switches consist of a / character followed immediately by a name. If the switch takes an argument, the argument is separated from the switch name by a colon (:) or equal sign (=). If the switch takes an argument and subarguments, each subargument is separated from the argument and from other subarguments by a colon. For example,

```
/NAME=JOB3
/VID="MY TAPE"
```

Switches accept arguments of standard types, such as decimal constant, alphanumeric string, and date-time.

**Table 22-3 File Specification Defaults**

| Condition | Default | Example |
|---|---|---|
| Name specified but no file type | Default assigned as appropriate to the current operation. For example, with the BATCH command BASIC, the default is .BAS. | ABC=ABC.type |
| Name, followed by dot, but no file type | Default file type is null. No file type is assigned. | ABC.=ABC |
| File type, but no name | Default filename is related to time of day. | .LIS=B2347P.LIS (created at 01:23:47 PM) |
| No file specification | Default filename (related to time of day) with default type as appropriate to current operation. | Null=B2347P.type |

Switch values can be negated by putting the characters NO between the / character and the switch name. For example,

/NOOBJ

This switch indicates that no object file is to be produced. Its most frequent use is in conjunction with the $BASIC and $COBOL commands.

**NOTE**

The negation characters NO are not considered part of the switch name. Thus, a negated switch must contain at least five characters. For example:

/NOOBJ or /NOOBJECT

is valid, but

/NOO

is invalid.

## 22.3 BATCH COMMANDS

The BATCH command set consists of

| | |
|---|---|
| $JOB | which begins a job |
| $EOJ | which ends a job |
| $BASIC | which executes BASIC-PLUS or BASIC-PLUS II compiler |
| <system command> | which executes a system utility function |
| $RUN | which executes a program |
| $DATA | which begins data images |

| | |
|---|---|
| $EOD | which ends data images |
| $MESSAGE | which logs message on operator services console |
| $MOUNT | which assigns a device |
| $DISMOUNT | which deassigns a device |
| $COBOL | which executes the COBOL compiler |
| $SORT | which executes the PDP-11 Sort program SORT11 |
| $FORTRAN | which executes the FORTRAN compiler |
| $DELETE | which deletes files |
| $COPY | which copies files |
| $PRINT | which queues a file for the default line printer |
| $DIRECTORY | which lists a file directory |
| $CREATE | which creates a file from data in the input stream |

### 22.3.1 $JOB

This command marks the beginning of a job. The following command switches are allowed.

| | |
|---|---|
| /NAME=jobname | This switch assigns a name to the job. Job names can be up to 6 characters long. This name overrides the control file name as the identifier of the job. |
| /NONAME | This switch indicates that no job name is defined. A default job name is assigned. The default job name is the name of the control file. The name appears in all messages to the system operator. |
| /LIMIT:nnn | This switch is used to assign an elapsed time limit to the job. The value of nnn, a decimal number, is interpreted as minutes. Note that the elapsed time taken to execute a job is heavily dependent on overall system loading. |
| /NOLIMIT | Gives the job an unlimited amount of elapsed time to complete. If neither /LIMIT:nn nor /NOLIMIT appear, the job is given 10 minutes elapsed time to complete execution before BATCH terminates it. |
| /CPU:nnn | This switch is used to assign a CPU time limit to the job. The value of nnn, a decimal number, is interpreted as seconds. If /CPU is specified, and /LIMIT is not specified, no elapsed time limit is enforced, and the only time limit is on CPU time. If both switches are specified, both limits are enforced. If no CPU time limit is specified, the allowable CPU time is infinite. |
| /NOCPU | Gives the job an unlimited amount of CPU time to complete. |
| /PRIORITY:n | Sets the RSTS/E job priority to n (or the next lowest multiple of 8) for the BATCH stream. For privileged users, n can be between -120 and +127; for non-privileged users, n is limited to a value betwen -120 and -8. Unless otherwise altered by the /PRIORITY:n switch, all jobs run at -8 priority. |

| | |
|---|---|
| /CCL | This switch allows the use of the system's interactive Concise Command Language. When this switch is specified, any of the system commands which do not conflict with existing Batch commands may follow the $ character. The batch processor ensures that the job is in the READY state before executing the command. |
| /ERROR:<operand> | This switch specifies the level of error which the BATCH processor should tolerate without terminating the job. The level is indicated by <operand>, which may be FAT[AL], WAR[NING], or NON[E]. If FATAL, all errors are tolerated until completion. If WARNING, a fatal error terminates the job, but warning errors are tolerated. If NONE, any error terminates the job. If a job is to be terminated because the error level has been exceeded, termination occurs when the job next asks for input. A message is entered in the log file giving the reason for termination. The default error level for the BATCH stream is determined at start-up time. |

The following specification field may be included:

| | |
|---|---|
| [n,m] | To have the job executed on an account other than that under which it was queued, a specification field may indicate the account number desired. This feature can be used only by a privileged user. |

The following error conditions are possible:

Unrecognized switch
Illegal switch value
Multiple conflicting specifications (switches)
Different account specified by non-privileged user
Higher priority desired by non-privileged user

### 22.3.2 $EOJ

This command marks the end of a job. The $EOJ command automatically dismounts all devices mounted by the job. $EOJ prints an appropriate message to the operator that the logical device should be dismounted. A logical deassignment is performed.

**NOTES**

1. The $EOJ command is implied when BATCH encounters a physical end-of-file condition or another $JOB control statement while processing a control file.
2. No switches are legal in the $EOJ command.

### 22.3.3 $BASIC

The $BASIC command calls a BASIC compiler, which compiles a source program. The format of the $BASIC command is:

```
$BASIC [switches] [specification fields [switches] ] [specification fields]
```

The following switches are valid in the command field:

| | |
|---|---|
| /BP1 | Use the BASIC-PLUS compiler. If neither /BP1 nor /BP2 appears, /BP1 is used. |
| /BP2 | Use the BASIC-PLUS II compiler. If neither /BP2 nor /BP1 appears, /BP1 is used. |

| | |
|---|---|
| /RUN | Execute (only) a previously compiled/task built program. If this switch is used, the entire command line must have one and only one file specification, and can have only one other switch: /EXECUTE. |
| /NORUN | Perform the compile/task build procedure, but do not execute the final file. |
| /OBJECT (legal only for BASIC-PLUS II runs; see /EXECUTE) | Create the object file filename.OBJ, where the filename is that of the source file. This switch implies /BP2, and therefore causes BASIC-PLUS II to be run; it also causes a task build operation. |
| /NOOBJECT | Create the object file filename.TMP, where the filename is that of the source file. Delete this .TMP (temporary) file upon completing the command. This switch implies /BP2, and therefore causes BASIC-PLUS II to be run. Thus, like /OBJECT, it is legal only in a BASIC-PLUS II run. |
| /LIST | Produce the listing file filename.LST, where the filename is that of the source file. If neither /LIST nor /NOLIST appears, /LIST is used. |
| /NOLIST | Do not produce a listing file. If neither /NOLIST nor /LIST appears, /LIST is used. |
| /MAP | Create the task builder map file filename.MAP, where the filename is that of the source file. This switch implies /BP2, and therefore causes BASIC-PLUS II to be run; it also causes a task build operation. Thus, it is legal only in a BASIC-PLUS II run. |
| /NOMAP | Do not create a map file. This switch implies /BP2, and therefore causes BASIC-PLUS II to be run. Thus, it is legal only in a BASIC-PLUS II run. |
| /EXECUTE (replaces /OBJECT for BASIC-PLUS runs) | Create an executable file, whose filename is that of the source file. Choose its type according to the following rules: If the language is BASIC-PLUS, give the file a .BAC type (filename.BAC). If the language is BASIC-PLUS II, give the file a .TSK type (filename.TSK). |
| /NOEXECUTE | If an executable file is not needed (as when /NORUN appears in the command), do not create one. If an executable file is needed, create a temporary one (.TMP), and delete it upon completion of $BASIC processing. |

One of the following switches may appear in the first specification field, described in the format guide at the start of this section.

| | |
|---|---|
| /SOURCE /BASIC | Both switches have the same meaning: i.e., that this is the BASIC-PLUS or BASIC-PLUS II source file on which to operate. |
| /EXECUTE | This switch is legal only if /RUN appears in the command field, and means that this is an executable file. |

In this field, any specification lacking a switch is assumed to be the input file for the command. Thus, only one file specification may appear without switches. This specification may not contain wild cards (neither asterisks nor question marks). If /NORUN appears in the command field, the lack of a switch here implies /BASIC. If /RUN appears in the command field, the lack of a switch here implies /EXECUTE. And if neither /RUN nor /NORUN appears in the command field, the lack of a switch implies /BASIC.

The optional specifications ending the format description (at the beginning of this section) define other files which may be needed in the operation. Any of the following switches may be used in the formats indicated. Each switch, however, may be used only once in the entire $BASIC command line. Moreover, its negation cannot be used anywhere in the command line (/NOLIST and /LIST, for example, cannot appear together in a command line).

| | |
|---|---|
| file specification/LIST | Define the listing file as specified. |
| file specification/OBJECT | Define the object file as specified (switch implies /BP2 and causes BASIC-PLUS II to be run, with task build; switch is legal only with BASIC-PLUS II). |
| file specification/MAP | Define the map file as specified (switch implies /BP2 and causes BASIC-PLUS II to be run, with task build; switch is legal only with BASIC-PLUS II). |
| file specification/EXECUTE | Define the executable file as specified. |

If BASIC-PLUS II is used and task build features are specified, multiple specifications of the following form may appear anywhere in the command line, delimited by spaces.

```
file specification/LIBRARY
```

Each such specification is a library file that will be linked with the BASIC-PLUS II program.

If a source file is not specified, the $BASIC command must be followed by a set of BASIC source statements, terminated by either $EOD (see Section 22.3.7) or some other recognized batch control statement. For example,

```
$BAS            LISTING/LIS

    BASIC
    Source
    Deck

$EOD
```

If a source file is explicitly specified, any source statements following this command are appended to the source program. Source statements following this command and having line numbers equal to those in the source program replace those in the source program. Source input must be provided, either through a file specification, or through source statements, or both.

If no listing file is specified, but the /LIST switch is present, the Batch processor creates, prints, and subsequently deletes the default listing file. If a file specification appears with the /LIST switch, the batch processor does not automatically queue the file for printing. To print the file specified as part of the batch job, the user must supply a $PRINT control statement described in Section 22.3.4.

If no executable file is specified with the /EXECUTE switch, a default executable file is created and is deleted after job completion. If an executable file is explicitly specified, it is preserved after job execution. Errors result from conflicting switch specifications such as both /BASIC and /SOURCE on different specification fields.

The default applied when a file is specified without a switch is /SOURCE.

The following error conditions are possible:

Unrecognized switch
Multiple conflicting specifications (switches)
File specification syntax error

### 22.3.4 Utility BATCH Commands

The following utility functions are provided by the RSTS/E batch processor.

| | |
|---|---|
| $DELETE | which deletes files |
| $COPY | which copies files |
| $PRINT | which prints a file on the system line printer by means of the line printer spooling program SPOOL |
| $DIRECTORY | which lists a file directory |
| $CREATE | which creates a file from data in the input stream |

**22.3.4.1 $DELETE** – The $DELETE command is used to delete specified files. It is issued in the following format.

```
$DELETE file1 [file2 . . . filen]
```

The filename and file type must be included. An asterisk is not valid in either the filename or the file type field.

No switches are used with $DELETE.

The following error conditions are possible:

No file specification
Syntax error in file specification

**22.3.4.2 $COPY** – $COPY is used to copy files. Use of the asterisk character in the file specification is invalid. The following are the valid switches.

| | |
|---|---|
| /OUTPUT | for new files to be created |
| /INPUT | for files to be copied |

The following is an example of the $COPY command.

```
$COPY TER.LIS/OUTPUT TERRY.LIS/INPUT
```

The $COPY command supports the use of + (plus sign) to indicate file concatenation. When used with $COPY. file concatenation results in the creation of a single file, consisting of files connected together. The + character appears in the file specification field, between the specifications of files to be concatenated. If no switch is specified, /INPUT is assumed.

The following error conditions are possible:

No output specification
No input specification
Multiple conflicting specifications
Syntax error in file description

**22.3.4.3 $PRINT** – The $PRINT command prints the contents of files on the system line printer by means of the spooling program SPOOL. File specifications accept all switches available in the QUE command (see Section 21.2). Asterisks can be used in file specifications. The $PRINT command is issued in the following format:

```
$PRINT file1/switches [file2 . . . filen]
```

The specification field contains the file or files to be printed.

The following error conditions are possible:

No file specification
Syntax error in file specification

**22.3.4.4 $DIRECTORY** – $DIRECTORY produces a directory listing of the file(s) in the specified account and is issued in the following format:

```
$DIRECTORY [specification field]
```

The specification field can contain file specifications. If no file specification appears, the $DIR command lists the contents of the current account on the batch log device. A file specification indicates the directory of a file or set of files and can contain an asterisk in either the filename field or file type field. For example,

```
$DIR *.BAS
```

This command creates a directory listing of all files in the current account with the .BAS type.

To create a directory in a disk file rather than on the batch log device, the user can specify a file and the /DIRECTORY switch. For example,

```
$DIR BAJOB.DIR/DIR
```

creates the directory listing in a file BAJOB.DIR on the system disk under the current account.

To create a directory in a disk file and to designate which files are to be listed, the user must specify both the /DIRECTORY and /INPUT switches with the related file specification. For example,

```
$DIR BA.DIR/DIR *.BAC/INPUT
```

The $DIR command shown subsequently creates a directory listing of all compiled BASIC-PLUS files and stores the listing in the file BA.DIR on the system disk under the current account.

The following error conditions are possible:

Syntax error in file specification
Multiple conflicting specifications

**22.3.4.5 $CREATE** – The $CREATE command creates a file as indicated in the specification field. The file consists of the data images following the $CREATE command in the input stream. Data images must follow $CREATE and must be terminated by $EOD, or an error occurs. The data images must not be preceded by any other command because the $CREATE function terminates on encountering a $ in the first column of a data image.

Any previously existing file of the name specified is deleted at batch execution time, and replaced by the file created by the $CREATE command.

The $CREATE command has the following format.

```
$CREATE file
```

The following error conditions are possible:

Syntax error in file specification
No file name specified
Non-comment characters following file specification

### 22.3.5 $RUN

The $RUN command causes execution of system programs. For example, to run PIP, the user types

```
$RUN $PIP
```

followed by appropriate PIP commands. The PIP program reads the commands as data images in the input stream. Execution of PIP is terminated when the next batch control statement is read.

No switches can be specified. The general format of $RUN is:

```
$RUN [file]
```

where file specifies the executable program. If file is omitted, the default current program is used.

The following error conditions are possible:

Syntax error in file specification
Non-comment characters following file specification

### 22.3.6 $DATA

The $DATA command provides a means of entering data to a program compiled and run by one of the language commands (e.g., $BASIC, $FORTRAN, $COBOL). $DATA ensures that the program will be run, unless the /NORUN switch was specified. It also ensures that if the program does not use all of its data, the remaining data will be flushed from the stream.

The $DATA command is issued without specification fields or switches, in the following format:

```
$DATA
```

### 22.3.7 $EOD

$EOD marks the end of data records included in the input stream following commands such as $BASIC, $CREATE, $DATA, and $RUN. For example,

```
$DATA
  .
  .
  .
data
  .
  .
  .
$EOD
```

### 22.3.8 $MESSAGE

The $MESSAGE command logs a message on the operator services console. It provides a way for the job to communicate with the operator. The command is issued in the following format:

```
$MESSAGE[/WAIT] message-string
```

The /WAIT switch can be used in the command field to indicate a pause to wait for operator action. The system pauses until the operator gives the appropriate command. For example, the following command halts the program until the operator takes action:

```
$MESSAGE/WAIT MOUNT SCRATCH TAPE ON DT0:
```

The WAIT condition remains in effect until the operator responds to the message on the operator services console.

### 22.3.9 $MOUNT

The $MOUNT command causes a mount message to be printed on the control console, and effects a logical to physical device assignment. The physical device refers to a physical device type. The operator responds with the device and unit number in the standard format (e.g., MT1:). An automatic /WAIT occurs. Logical device names of up to six characters are used to specify logical devices.

The $MOUNT command is issued in the following format:

```
$MOUNT devn: [/switch] devm: [/switch]
```

Both the logical device and the physical device must be specified. The colon is required as the terminator for each device specification. The following switches can be used for the physical device.

| | |
|---|---|
| /PHYSICAL | identifies the device specification to be the physical device (default) |
| /WRITE | tells the operator to write-enable the device (or volume) |
| /NOWRITE | tells the operator to write-protect the volume |
| /VID:<string> | <string> is a visual identification which identifies the volume for the operator |
| /DEN[SITY]:nnn | specifies density for magtape |
| /PAR[ITY]:[ODD] [EVEN] | specifies odd or even parity for magtape |

The following switch is used with the logical device:

| | |
|---|---|
| /LOGICAL | identifies the device specification to be the logical device name; this specification must correspond to the PACK ID for RSTS/E disks. |

The /VID switch on the physical device field is used to specify the volume identification. The value associated with /VID is the name physically attached to the volume. It is included to help the operator locate the volume.

If the name specified with /VID must contain blanks, it can be delimited by quotes. The following example illustrates:

```
/VID="FJM JT"
```

No blanks are allowed in a string not delimited by quotes. For example,

```
$MOU M7:/PHY/VID="MY TAPE" TAPE:/LOG
```

In this example, logical device name TAPE is assigned to a 7-track magnetic tape unit. The operator is told that the reel of tape to be physically mounted is labeled MY TAPE. He then responds with the device and unit number on which the tape is mounted. Thereafter, in the batch command file, reference to the device TAPE: accesses the physical device on which the operator mounted the reel MY TAPE. If the physical device is a removable disk pack or cartridge, the logical device name must be the pack identification. The batch processor logically mounts and unlocks private disks which the operator mounts as a result of $MOUNT.

The valid physical devices that can be requested for mounting are:

| | |
|---|---|
| CR: | Card Reader |
| DK: | Disk Cartridge |
| DP: | RP11-C/RP03 or RP02 disk pack |
| DB: | RH11/RP04 disk pack |
| DM: | RK06 disk pack |
| DX: | RX01 floppy disk |
| DT: | DECtape |
| LL: | Line printer with lower case |
| LP: | Line printer |
| LU: | Line printer with Upper case only |
| M7: | 7-track Magtape |
| M9: | 9-track Magtape |
| MT: | TM11/TU10 or TS03 magtape |
| MM: | TM02/TU16 or TU45 magtape |
| PP: | Paper tape punch |
| PR: | Paper tape reader |
| PS: | Public Storage (equivalent to SY:) |
| SY: | System device |
| TT: | Teletype (or terminal) |

The following error conditions are possible:

Syntax error in device specification fields
Invalid device name/unit
Invalid logical device name specifications
Unit number already assigned
Both physical and logical names have not been specified

### 22.3.10 $DISMOUNT

The $DISMOUNT command causes the logical to physical device assignment effected by the $MOUNT command to be nullified. It also prints an operator message, requesting that the volume be dismounted. If a /WAIT switch is included in the command field, the job will not resume until a response, as with the $MESSAGE command, is received from the operator. For example,

```
$DIS/WAI TAPE:
```

sends a message to the operator to dismount the magtape that was mounted by the example in Section 22.3.9. As a result of the switch /WAI, BATCH pauses until the operator responds.

All devices are automatically dismounted at end-of-job (EOJ).

The following error conditions are possible:

Syntax error in specification field
Illegal switches
Logical device not assigned

### 22.3.11 $COBOL

The $COBOL command calls the COBOL compiler which compiles the source program and generates an object program. The format of the command is as follows.

```
$COBOL[switches] [specification fields [switches] ]
```

The following command field switches are valid.

| | |
|---|---|
| /RUN | Execute the previously compiled object file. Only an object file can be specified. |
| /NORUN | Compile the source program but do not execute the object. If /NORUN is omitted, the source program is compiled and executed. |
| /OBJECT | Produce a compiled file. If neither /OBJECT nor /NOOBJECT appears, /OBJECT is used. |
| /NOOBJECT | Do not produce an object file. If neither /OBJECT nor /NOOBJECT appears, /OBJECT is used. |
| /LIST | Produce a listing file. If neither /LIST nor /NOLIST appears, /LIST is used. |
| /NOLIST | Do not produce a listing file. If neither /LIST nor /NOLIST appears, /LIST is used. |
| /MAP | Include the DATA division map in the listing file. |
| /LOD | Create the file nnnnnn.MAP (where nnnnnn is the source file name) to contain the program load map. |
| /CVF | Source code is in conventional format. |
| /ACC:n | Accept errors in the source code of severity n or less. |
| /ERR:n | Suppress the printing of diagnostic messages if error severity is less than n. |
| /USW:n:n . . . | Set run time user switches for the compiler. Switch values must be separated by colons. The range of values is 1 through 16. |
| /HELP | Print a help message in the log. No other switches and no file specifications are permitted with the /HELP switch. |

If neither /RUN nor /NORUN appears in the command field, COBOL automatically compiles the source program and executes the object program. If either a source file or a listing file or both are specified with /RUN, a conflict occurs. The /RUN switch indicates that an object file is to be executed. Source and listing files are specified only when a compilation is performed. BATCH does not report the file specification(s) as errors but does not pass them to the COBOL compiler for processing.

A maximum of 3 file specifications can appear in the specification field. Each specification can be differentiated by one of the following switches.

| | |
|---|---|
| /COBOL or /SOURCE | Indicates input source file. If specification has no switch, /COBOL is used for that file. |
| /OBJECT | Indicates output (compiled) file. |
| /LIST | Indicates output listing file. |

If /OBJECT does not appear in the specification field, BATCH creates a default object file and deletes it after the program run is completed. If a listing file does not appear in the specification field, but the /LIST switch is included, a default listing file is created, printed, and deleted. Explicitly specified output files (both object and listing) survive the execution of the batch job that created them. If a listing file appears in the specification field, BATCH creates the listing file but does not automatically queue it for printing. To print the listing file as part of the BATCH run, a $PRINT control statement must be supplied. (Section 22.3.4 describes $PRINT.)

If a file specification appears without a switch, BATCH uses the /COBOL switch for that file.

If a source input file specification does not appear in the $COBOL control statement, source statements must immediately follow and must be terminated by either an $EOD statement or some other BATCH control statement. For example,

```
    $COBOL/MAP/LOD COB.LST/LIS

          COBOL
          Source
          Deck

    $EOD
```

Because the command field contains neither /RUN nor /NORUN, BATCH assumes a compilation and execution is to be done. A source file is not specified and the data following the $COBOL command is compiled.

The following command and text describe the defaults used in the $COBOL command.

```
    $COBOL FILE
```

A compilation and execution is done. Because the specification FILE contains no switch, /COBOL is used. The default type .CBL is used. The source program FILE.CBL is compiled and a default object file is created and executed. The object file is automatically deleted before the next command is executed with one exception. If the next command is $DATA, the object file is deleted after the end of data and before the command following the data. A default listing file is created and automatically queued for printing. The listing file is deleted after it is printed.

The following error conditions are possible:

Unrecognized switch
Multiple conflicting switches
File specification syntax error
No source input (neither file nor source statements)

### 22.3.12 $SORT

The $SORT control statement runs the PDP-11 Sort program SORT11 (not the RSTS/E program SORT.BAC). The SORT11 program is on RSTS/E systems with the COBOL compiler. For more information on the PDP-11 program SORT11, refer to the *PDP-11 SORT Reference Manual.*

The format of the $SORT control statement is as follows.

```
    $SORT[switches]  [file specification[switches]]
```

The following are the valid switches for the command field.

| | |
|---|---|
| /SIZE:n | Use n as maximum record size in bytes. Size can be between 1 and 16383. |
| /FILES:n | Use n scratch files; n can be between 3 and 10. |

| | |
|---|---|
| /PROCESS:x | Use the sort process given by x. Values can be R (record sort), T (tag sort), A (ADDROUT sort), or I (index sort). If /PROCESS:x does not appear or if :x is omitted from the /PROCESS:x switch, a record sort is performed. |
| /KEYS:abm.n | Use the sorting key field defined by entries for a, b, m and n. Maximum of 10 keys can be specified if each is separated by a colon as follows.<br><br>/KEYS:abm.n:abm.n: . . . abm.n:abm.n |
| /RIN | Specifies the input file as a COBOL relative file. Without /RIN, input file is assumed to be a sequential ASCII file. |
| /ROUT | Specifies the output file as a COBOL relative file. Without /ROUT, the output file is created as a sequential ASCII file. |

The command field of the $SORT control statement must have the /SIZE:n switch to define the record size. The requirements for other command field switches depend on file specifications present and the type of sort requested.

A maximum of three file specifications can appear in the $SORT control statement: an input file, an output file, and a specification file. To distinguish these files, the following switches are used.

| | |
|---|---|
| /INPUT | the file to be sorted |
| /OUTPUT | the file to contain the sorted data |
| /SPECIFICATION | the file which contains the control information for the sorting process. |

A file specification without a switch is used as the file to be sorted. If the /SPECIFICATION switch is used, the /KEYS and /PROCESS switches must not appear in the command field. If a specification file is not given in the control statement, the /KEYS switch must be included in the command field to control the sorting process.

Missing elements in a file specification are replaced by BATCH default elements. If type is omitted from the file specification, BATCH uses .SRT as the type.

The following is a sample batch stream using $SORT commands.

```
$JOB/NAME=SRT002/LIMIT=30
$SORT/PRO:T/SIZ:100/KEY:04.1:01.1 F100.100 A/OUT
$SORT/PRO:T/RIN/KEY:1.4/SIZ:100 R200.100/INP B/OUT
$SORT/SIZ:100/ROU/KEY:1.4/PRO:T F100.100/INP X/OUT
$SORT/SIZ:60/KEY:3.5/FIL:3 V100.060/INP C/OUT
$SORT/SIZ:100/RIN R200.100/INP D/OUT SPEC.001/SPE
```

The /SIZE switch appears in each command to define the record size in the file to be sorted. In the first $SORT control statement, file F100.100 is used as the input file. In the second statement, the /RIN switch in the command field denotes the input file R200.100 as a COBOL relative file. In the third statement, the /ROUT switch causes the output file X.SRT to be a COBOL relative file. For the fourth $SORT control statement, a record sort is performed on file V100.060 because the /PROCESS:x switch is absent. For the fifth $SORT statement, the sorting process is controlled by data in the specification file SPEC.001. The /PROCESS:x and /KEYS switches are not permitted. For all files without a type in the specification, BATCH uses the default .SRT.

### 22.3.13 $FORTRAN

The $FORTRAN command calls the FORTRAN compiler, which compiles the source program and generates an object program. The format of the command is

```
$FOR[TRAN] [switches] [specification field[switch] ] [spec fields[switch] ]
```

The following switches are valid in the command field:

| | |
|---|---|
| /RUN | Execute the previously compiled file. Only an object file can be specified. |
| /NORUN | Compile the source program but do not execute the object file. |
| /OBJECT | Create the compiled file filename.OBJ, where the filename is that of the source file. If neither /OBJECT nor /NOOBJECT appears, /OBJECT is used. |
| /NOOBJECT | Do not create an object file. If neither /NOOBJECT nor /OBJECT appears, /OBJECT is used. |
| /LIST | Produce the listing file filename.LST, where the filename is that of the source file. If neither /LIST nor /NOLIST appears, /LIST is used. |
| /NOLIST | Do not produce a listing file. If neither /NOLIST nor /LIST appears, /LIST is used. |
| /MAP | Create the map file filename.MAP, where the filename is that of the source file. |
| /NOMAP | Do not create a map file. |

One of the following switches may appear in the first specification field described in the format guide at the start of this section.

| | |
|---|---|
| /FORTRAN<br>/SOURCE | Both switches have the same meaning: i.e., that this is the source file on which to operate. If a file specification lacks a switch, this meaning is applied to it. |

The optional specifications ending the format description define other files which may be needed in the operation. Any of the following switches may be used in the formats indicated. Each switch, however, may be used only once in the entire $FORTRAN command line. Moreover, its negation cannot be used anywhere in the command line (/NOLIST and /LIST, for example, cannot appear together in a command line).

| | |
|---|---|
| file specification/LIST | Define the listing file as specified. |
| file specification/OBJECT | Define the object file as specified. |
| file specification/MAP | Define the map file as specified. |

Multiple specifications of the following form may appear anywhere in the command line, delimited by spaces:

```
file specification/LIBRARY
```

Each such specification is a library file that will be linked with the FORTRAN program.

## 22.4 BATCH OPERATING PROCEDURES

This section describes how the RSTS/E system user requests batch processing and how the batch processor generates output.

### 22.4.1 Requesting a Batch Job Run

To request the running of a batch job, the user runs the library program QUE and specifies the batch control file or files as follows.

```
RUN $QUE
QUE      V06B-03 RSTS V06B-02 Timesharing
#Q BA:BATJOB=FILE1,FILE2,FILE3.DAT
#
```

The user normally queues a batch job to device BA:. The job and log files in this example will be named BATJOB, and the files FILE1.CTL, FILE2.CTL, and FILE3.DAT will be concatenated to form the batch control file. The log file BATJOB.LOG will be printed after the job is complete.

The CCL command QUEUE, if available on the system, may also be used to submit a job for batch processing.

### 22.4.2 Batch Processing

As the batch control file is read, it is checked for command sequence and syntactical validity. If an error is detected, an error message is printed in the log file. The job will not be run, but syntax checking will continue through the remainder of the file(s).

A $MESSAGE/WAIT, a $MOUNT, or a $DISMOUNT/WAIT will cause the job to pause for an operator response. Until the operator takes action, no further commands will be sent to the pseudo keyboard.

If no errors are detected, the job is processed. A log is created, showing the sequence of Batch commands processed during the course of the job. If program output is directed to KB:, this output appears following the command that caused the program to execute. In the example that follows, a BATCH job named JOB1 has been run. The Batch control file contained the following sequence of commands:

```
$JOB/NAME=JOB1/LIMIT=4
$CREATE SUB1.BAS

     source statements

$EOD
$BASIC/BP2 LISTING/LIS MAIN/OBJ

     source statements

$DATA

     data

$PRINT SUB1.BAS
$EOJ
```

These commands have the following effect:

```
$JOB/NAME=JOB1/LIMIT=4
```

A job name of JOB1 is assigned to the job. This name appears on the job log along with the time and date of the job's execution. A time limit of four minutes is set. If the job is not finished in four minutes from its start (actual elapsed time), the job is terminated, and the appropriate error message is printed in the log.

```
$CREATE SUB1.BAS
```

A BASIC source file named SUB1 is created, from data records which must follow the $CREATE command.

```
$EOD
```

The $EOD command signals the end of SUB1.BAS.

```
$BASIC/BP2 LISTING/LIS MAIN/OBJ
```

The source statements following this command are compiled by the BASIC-PLUS II compiler. A listing of the source statements is created in the file LISTIN.B2S, and the object data is placed in the file MAIN.OBJ. The temporary task built file has the extension .EXE. This file is executed.

```
$DATA
```

The data to be read during execution of MAIN.BAC follows this command.

```
$PRINT SUB1.BAS
```

The source file created by $CREATE SUB1.BAS is printed. This command also has the effect of terminating data input to MAIN.BAC.

```
$EOJ
```

This command signals the end of job JOB1.

## 22.4.3 Error Procedures

When a syntax error is detected in a batch command, the job is not executed. Instead, an error log is printed listing all commands and data scanned along with the appropriate error message(s). The batch log file always indicates all command lines scanned. If an error is found on a command line, the error message follows the command, marked with question marks (????????????). Scanning of the control file continues, but the job will not be executed.

If no syntax errors occur, the time of output of lines will be indicated in the left margin of the log. All normal terminal interaction corresponding to the BATCH commands will appear in the log. Table 22-4 lists the BATCH error messages and their meanings.

**Table 22-4 Summary of BATCH Error Messages**

| Message | Meaning |
|---|---|
| BATCH Being Shut Down | The BATCH processor is going off-line and the user job must be terminated. |
| Cannot Use That Account | An account specification appeared in the $JOB command but the user who queued the request was not privileged. |
| Cannot Increase Priority | A /PRIORITY:n switch appeared in the $JOB command. The user was privileged but specified a value for n greater than 127. Or, the user was nonprivileged and specified a value greater than -8. |
| Continuation Missing | The dash (-) character was the last non-blank character in a control statement to continue the statement on the next line, but the following line did not begin with a dollar sign ($) and a blank. |
| Device not MOUNTed | A $DISMOUNT command was present but the device indicated had not been mounted. |
| Disk Mount Failure | The volume to be mounted was not correct (pack IDs did not match) or the device was in use by another job. |

(continued on next page)

**Table 22-4 (Cont.) Summary of BATCH Error Messages**

| Message | Meaning |
|---|---|
| Invalid Command | An undefined command name followed the $ character in a statement but the /CCL command switch had not been specified in the $JOB command. |
| Invalid Specification Field | The specification given in a control statement is in the wrong format. |
| Invalid Switch | The switch used in the command field or in the specification field is undefined, in the wrong format, or is privileged. |
| No BATCH Jobs Possible At This Time | The BATCH processor requires a pseudo keyboard to execute a job but one is not available. The user must requeue his request. |
| No Such Account | The account specified in the $JOB command or in a specification field could not be found on the device. |
| SEQUENCE Not Supported Yet | The $SEQUENCE command is not available with this version of BATCH. |
| Time Limit Exceeded | Time specified in $JOB command is insufficient to execute the job. The user should specify a larger limit by using /LIMIT=nnn or /NOLIMIT switch. |
| Too Many Mounted Devices | The job has requested mounting of more devices than the maximum (12) allowed by BATCH. |
| Unable To Log In BATCH Job | To execute a user request, the BATCH processor logs a job into the system using the account under which the job was queued or the account specified in the $JOB command. For some reason, the login procedure failed. For example, logins had been disabled. The job will be requeued for later execution. |
| Unmatched Parentheses | An opening left parenthesis appears in a specification field but an accompanying closing parenthesis is not found. |
| Unmatched Quotation Marks | Quotation marks (and single quotation marks) must be paired in a control statement. |

# PART V

# RSTS/E PERIPHERAL DEVICES

# CHAPTER 23
# RSTS/E PERIPHERAL DEVICES

RSTS/E has several peripherals which are available to the user. These devices include:

user terminal (ASR-33, ASR-35, LA30 & LA36 DECwriters, VT05 & VT50 displays)

high-speed paper tape reader/punch

card reader

line printer

DECtape

magtape

floppy disk

While normal operation of a computer system is by programmed control, manual operation is necessary for some tasks. This chapter describes the manual control and operation of the common RSTS/E user peripherals.

Table 23-1 lists the RSTS/E peripherals according to their locations in Chapter 23.

**Table 23-1 A Guide to the RSTS/E Peripheral Devices**

| Device | Location |
|---|---|
| Card Reader (CR11) | 23.3 |
| DECtape Control & Transport | 23.5 |
| Floppy Disk (RX11) | 23.10 |
| Line Printer (LP11) | 23.4 |
| Magtape Control & Transport (TM11/TU10 and TJU16) | 23.6 |
| Paper Tape Punch | |
| Low-Speed (ASR-33 Teletype)[1] | 23.1.5 |
| High-Speed | 23.2.2 |
| Paper Tape Reader | |
| Low-Speed (ASR-33 Teletype) | 23.1.4 |
| High-Speed | 23.2.2 |

(continued on next page)

[1] Teletype is a trademark of Teletype Corporation.

**Table 23-1 (Cont.) A Guide to the RSTS/E Peripheral Devices**

| Device | Location |
|---|---|
| Terminals | |
| DECwriter II (LA36) | 23.9 |
| Teletype (ASR-33) | 23.1 |
| VT05 Display Terminal | 23.7 |
| 2741 Communications Terminal | 23.8 |

## 23.1 ASR-33 TELETYPE

The ASR-33 Teletype is an inexpensive, commonly employed user terminal. Major features are noted in Figure 23-1.

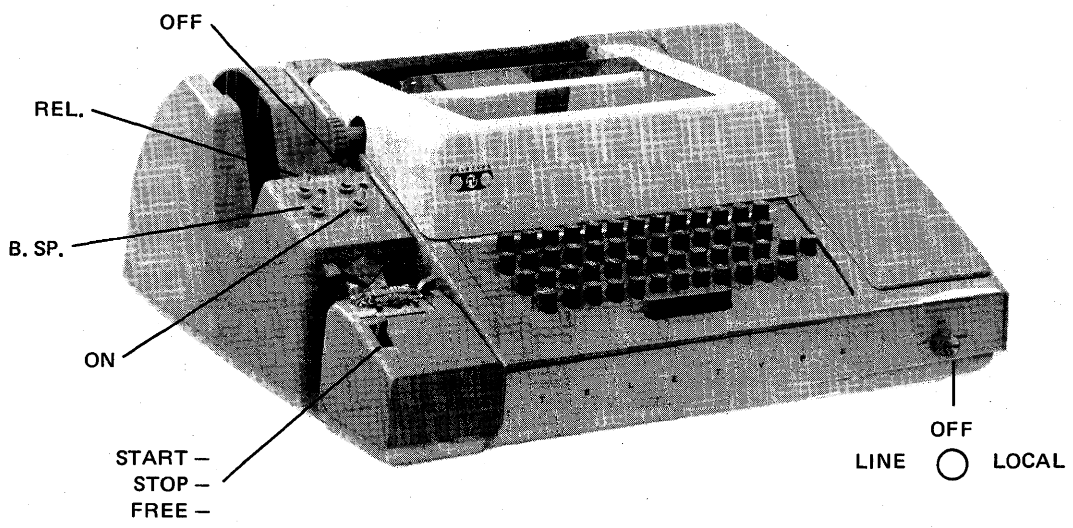


Figure 23-1 ASR-33 Teletype Console

The components of the Teletype unit and their functions are described below.

### 23.1.1 Control Knob

The control knob of the ASR-33 Teletype console has three positions:

| | |
|---|---|
| LINE | The console has power and is connected to the system as an I/O device under RSTS/E control. |
| OFF | The console does not have power. |
| LOCAL | The console has power for off-line operation under control of the keyboard and switches only. |

### 23.1.2 Keyboard

The Teletype keyboard shown in Figure 23-2 is similar to a typewriter keyboard, except that some nonprinting characters are included as upper case elements. For typing characters or symbols such as $, %, or # which appear on the upper portion of numeric keys and some alphabetic keys, the SHIFT key is depressed while the desired key is typed.

Nonprinting operational functions are shown on the upper part of some alphabetic keys. By depressing the CTRL (control) key and typing the desired key, these functions are activated.[1]

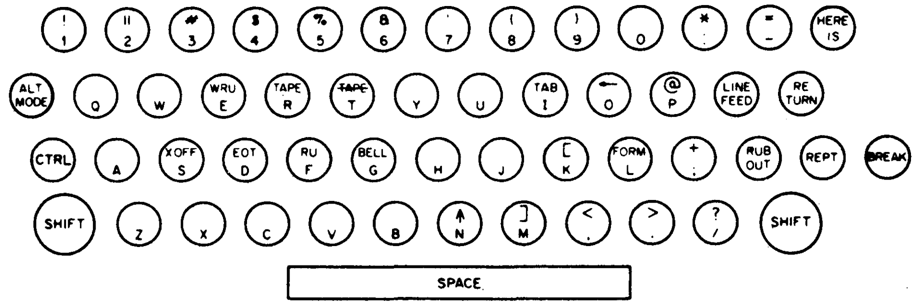


Figure 23-2 Teletype Keyboard

### 23.1.3 Printer

The Teletype printer provides a printed copy of input and output at ten characters per second, maximum rate. When the Teletype unit is on-line to the system (control knob turned to LINE), the copy is generated by the computer (even the echoing of the characters typed by the user); when the Teletype unit is off-line (control knob turned to LOCAL), the copy is generated directly from the keyboard onto the printer as a key is struck.

### 23.1.4 Low-Speed Paper Tape Reader

The paper tape reader is used to read data punched on eight-channel perforated paper tape at a rate of 10 characters per second, maximum. The reader controls are shown in Figure 23-1 and described below.

| | |
|---|---|
| START | The reader is activated; reader sprocket wheel is engaged and operative. |
| STOP | The reader is deactivated; reader sprocket wheel is engaged but not operative. |
| FREE | The reader is deactivated; reader sprocket wheel is disengaged. |

The following procedure describes how to properly position paper tape in the low-speed reader.

1. Raise the tape retainer cover.
2. Set reader control to FREE.
3. Position the leader portion of the tape over the read pins with the sprocket (feed) holes over the sprocket (feed) wheel and with the arrow on the tape (printed or cut) pointing outward (forward).
4. Close the tape retainer cover.
5. Make sure that the tape moves freely (if the tape does not move back and forth freely, the paper feed holes are not properly positioned).
6. Set reader control to START, and the tape is ready.

### 23.1.5 Low-Speed Paper Tape Punch

The paper tape punch is used to perforate eight-channel rolled oiled paper tape at a maximum rate of 10 characters per second. The punch controls are shown in Figure 23-1 and described below.

| | |
|---|---|
| RELease | Disengages the tape to allow tape removal or loading. |
| B.SP | Backspaces the tape one space for each firm depression of the B.SP button. |

[1] Although not shown on most keyboards, SHIFT/L produces the backslash character (\) and SHIFT/K and SHIFT/M produce the square brackets, [ and ], respectively.

| | |
|---|---|
| ON (LOCK ON) | Activates the punch. |
| OFF (UNLOCK) | Deactivates the punch. |

Blank leader/trailer tape is generated by:

1. Turning the control knob to LOCAL.
2. Turning the punch control to ON.
3. Typing the HERE IS key.
4. Turning the punch control to OFF.
5. Turning the control knob to LINE.

## 23.2 HIGH-SPEED PAPER TAPE READER AND PUNCH UNITS

One high-speed paper tape unit can be provided with each RSTS/E system. This unit is mounted on the central computer console. A high-speed paper tape unit is pictured in Figure 23-3 and descriptions of the reader and punch units follow.

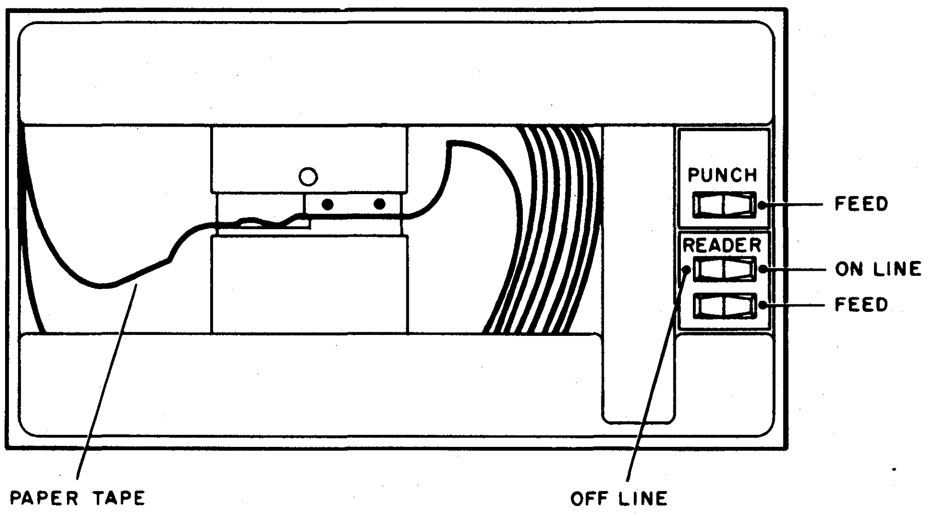


Figure 23-3 High-Speed Paper Tape Reader/Punch

### 23.2.1 High-Speed Reader Unit

The high-speed paper tape reader is used to read data from eight-channel, fan-folded (non-oiled), perforated paper tape photoelectrically at a rate of 300 characters per second, maximum.

**NOTE**

Tape from the Teletype punch should not be used with the high-speed reader as the oil on the tape causes lint and dust to collect on the photoelectric cells.

Primary power is applied to the reader when the computer power is on.

In order to use the high-speed reader as an input device, turn the reader ON LINE/OFF LINE rocker switch to ON LINE. Load tape into the reader as explained below:

1. Raise the tape retainer cover.
2. Place tape in right-hand bin with printed arrows pointing toward left-hand bin. (Channel one of the tape is toward the rear of the bin.)

3. Place several folds of blank tape past the reader and into the left-hand bin.
4. Place the tape over the reader head with feed holes engaged in the teeth of the sprocket wheel.
5. Close the tape retainer cover.
6. Depress the FEED rocker switch until leader tape is over the reader head.

The reader is capable of sensing whether a tape is in the reader. If an attempt is made to read a tape when the reader is either empty or OFF LINE, an error is generated.

### 23.2.2 High-Speed Punch Unit

The high-speed paper tape punch is used to record computer output on eight-channel, fan-folded, non-oiled paper tape at a rate of 50 characters per second maximum. All characters are punched under program control from the computer. Blank tape (feed holes only, no data), is produced by pressing the punch FEED rocker switch. The punch unit has power turned on whenever the computer has power; it does not require any additional on/off switch.

Fan-folded paper tape is generally grey in color. When a box of tape is nearly empty, purple tape is produced. Rather than risk running out of tape while punching, replace the box of paper tape at this point or notify the system manager who will replace the tape.

## 23.3 CR11 CARD READER

The CR11 card reader allows the RSTS/E system to accept information from punched 80 column data cards. Cards can be read under program control at rates of up to 200 or 300 cards per minute.

Power to the reader, shown in Figure 23-4, is controlled by the ON/OFF switch on the upper left hand corner of the back panel. Two toggle switches are present on the back panel and should be set to AUTO and REMOTE for proper operation. The LAMP TEST button on the reader back panel can be depressed to check the operation of the various reader lights on the front panel.

In order to use the card reader:

1. Remove card weight from input hopper. Place cards loosely in input hopper. The first card to be read is placed at the front of the deck, "9" edge down, column 1 to the left. Replace card weight on top of cards in input hopper. Cards should not be packed tightly.
2. Press green RESET button. Wait 4 seconds for RESET light to come on. The card deck is now able to be read under program control.
3. Cards may be loaded while the reader is operating provided tension is maintained on the front of the deck as cards are added to the rear. Additional cards should not be loaded until the hopper is 1/2 to 2/3 empty.
4. The output stacker bin can be unloaded while cards are being read. Care should be taken to maintain the order of the deck.

The various lights and switches (buttons) on the reader front panel and their significance are described in Table 23-2.

## 23.4 LP11 LINE PRINTER

The LP11 line printer, pictured in Figure 23-5, has an 80 column capacity, prints at a rate of 356 lines per minute at a full 80 columns, and can print 1100 lines per minute at 20 columns. These rates are based on a 64-character set. A 96-character set and 132-column version are also available. The print rate is dependent upon the data and number of columns to be printed.

Characters are loaded into a 20-character printer memory via a Line Printer Buffer. When this buffer is full, the characters are automatically printed. This process continues until the 80 columns (four print zones) have been printed or a carriage return, line feed, or form feed character is recognized. The printer responds only to codes representing the character set and three control characters. All other codes are ignored.

**Table 23-2 Card Reader Controls**

| Light/Switch | Function |
|---|---|
| POWER | light indicates that there is power to the reader. |
| READ CHECK | light indicates a reading error, torn card, or card too long for reader. Reader stops and RESET light is out. |
| PICK CHECK | light indicates inability to remove card from input hopper. Reader stops and RESET light is out. |
| STACK CHECK | light indicates inability to remove card from input hopper. Reader stops and RESET light is out. |
| HOPPER CHECK | light indicates that there are no cards in input hopper or the output stacker is full. Condition must be manually corrected to allow further operation. |
| STOP | button, when depressed, causes red light to go on momentarily. RESET light goes out and reader operation stops as soon as the card currently in the read station has been read. |
| RESET | button, when depressed, causes green RESET light to go on and initializes card reader logic. |

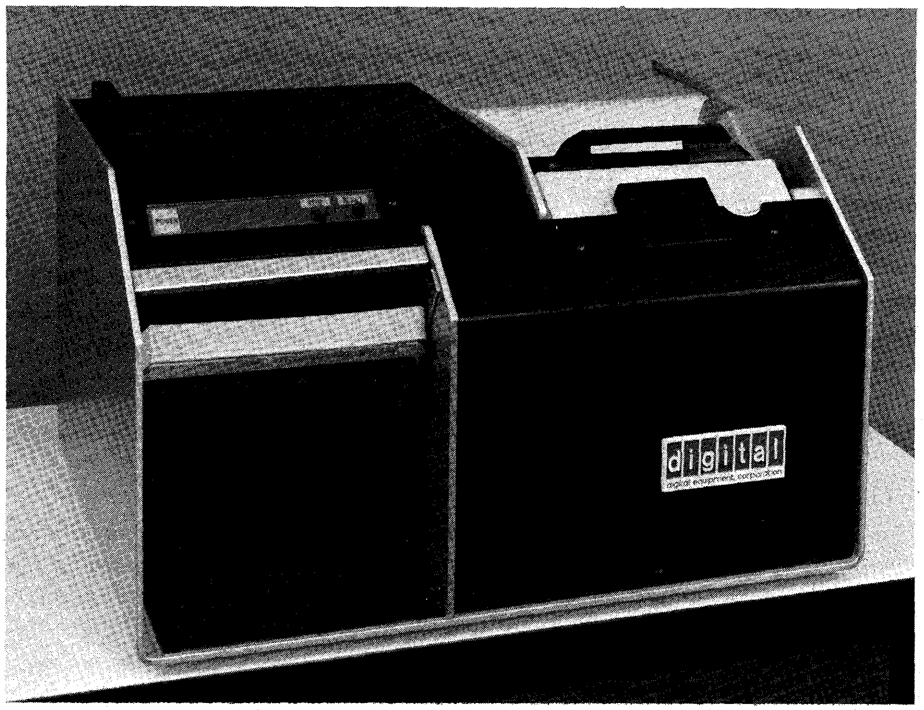


Figure 23-4 CR11 Punched Card Reader

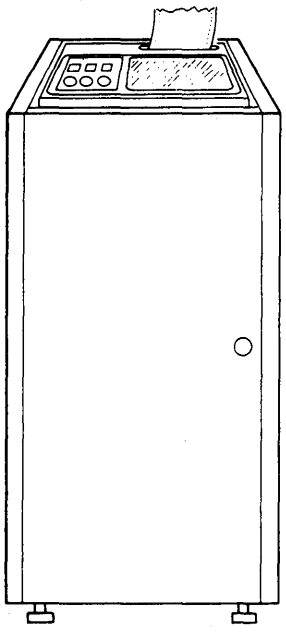


Figure 23-5 LP11 Line Printer System (80-column model)

### 23.4.1 Line Printer Character Set

The 64-character set consists of the 26 upper case letters (A-Z), ten numerals (0-9), 27 special characters and the space character. The 96-character set contains all of the above plus 26 lower case letters and 6 additional symbols. The character codes are 7-bit ASCII.

Characters are printed 10 characters per inch and 6 lines per inch.

Line printers can use paper varying in width from 4 inches to 9-7/8 inches for the 80-column printer. Forms making up to six copies can be used when multiple copy printing is desired.

The special symbols available are as follows:

| 64-character set | 96-character set |
|---|---|
| ! " # $ % & ' ( ) | all of the 64-character |
| * + , - . / ; : < | set symbols |
| > = ? @ \ [ ] ↑ | DEL |

ASCII numeric equivalents for the various characters are contained in Appendix D.

### 23.4.2 Line Printer Operation

Figure 23-6 illustrates the line printer control panel on which are mounted three indicator lights and three toggle switches. Operation of these switches and the power switch, and the meaning of the lights is explained in Table 23-3.

**Table 23-3 Line Printer Controls**

| Light/Switch | Function |
|---|---|
| POWER light | Glows red to indicate main power switch (located inside cabinet) is at ON position and power is available to the printer. |
| READY light | Glows white, shortly after the POWER light goes on to indicate that internal components have reached synchronous state, paper is loaded, and the printer is ready to operate. |
| ON LINE light | Glows white to indicate that ON LINE/OFF LINE toggle switch is in ON LINE position. |
| ON LINE/OFF LINE switch | This three-position toggle switch is spring-returned to center. When momentarily positioned at ON LINE it logically connects the printer to the computer and causes the ON LINE light to glow. Positioned momentarily at OFF LINE, the logical connection to the computer is broken, the ON LINE light goes off, and the TOP OF FORM and PAPER STEP switches are enabled. If printer is switched to OFF LINE, the ON LINE light remains on until either PAPER STEP or TOP OF FORM switch is activated. The printer should again be turned ON LINE. |
| TOP OF FORM switch | This two-position toggle switch is tipped toward the rear of the cabinet to roll the form to the top of the succeeding page. It is spring-returned to center position, and produces a single top-of-form operation each time it is actuated. The switch is effective when the printer is off line. |
| PAPER STEP switch | This two-position toggle switch operates similarly to TOP OF FORM but produces a single line step each time it is actuated. It is only effective when the printer is off line. |
| ON/OFF (main power) switch | This switch controls line current to the printer. To gain access to it, the printer front panel is unlatched, by pushing the circular button on the right hand edge, and opened to the left on its hinges. The switch is located to the left of center approximately fourteen inches below the top. If power is available, the red POWER light on the control panel will glow when the switch is positioned at ON.<br><br>The switch is on when in the up position. The ON and OFF labels are printed on the stem of the switch. A group of two switches and three indicator lights, above the main power switch, are for the use of technicians in making initial adjustments to the printer. |

The following procedure is used when loading paper in the line printer.

1. Open front door of cabinet. POWER light should be on. Printer should be off line.
2. Lift control panel TOP OF FORM switch and release to move tractors to correct loading position.
3. Open drum gate by moving drum gate latch knob to left and up. Swing drum gate open.
4. Adjust right hand tractor paper width adjustment for proper paper width if necessary. (Loosen set screw on 80 column printer; user release mechanism on 132 column printer.) Tighten tractor after adjustment.
5. Open spring loaded pressure plates on both tractors.

6. Load paper so that the two red arrows point to a perforation. Paper should lie smoothly between tractors without wrinkling or tearing the feed holes.
7. Close spring-loaded pressure plates on both tractors.
8. Adjust COPIES CONTROL lever to proper number of copies to be made, if necessary. Set to 1 or 2 for single forms, set to 5 or 6 for six-part forms.
9. Close drum gate and lock in position with drum gate latch. After 10 seconds the READY indicator should light.
10. Lift TOP OF FORM switch several times to ensure paper is feeding properly.
11. Set printer to on line. ON LINE indicator should light. At this point printed matter can be aligned with the paper lines, if desired, by rotating the paper vertical adjustment knob.

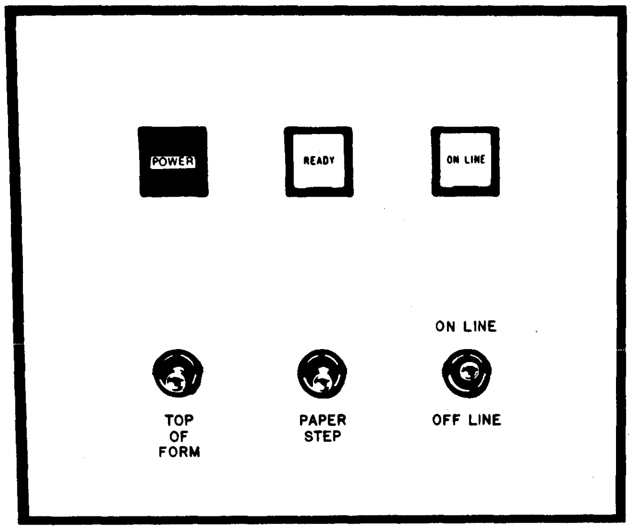


Figure 23-6 Line Printer Control Panel

## 23.5 TC11/TU56 DECTAPE CONTROL AND TRANSPORT

DECtape units are available on most RSTS/E systems. DECtape serves as an auxiliary magnetic tape storage facility to the system disk(s).

A DECtape peripheral unit consists of three components:

1. TU56 DECtape transport, pictured in Figure 23-7, which reads and/or writes information on magnetic tape.
2. TC11 Controller, the interface which controls information transfer. One controller serves up to four transports (up to 8 tape drives). The Controller is transparent to the RSTS/E user.
3. DECtape, the recording medium used for data storage, consists or reel mounted magnetic tape formatted to permit read/write operations in either direction, error checking, block identification, and timing control.

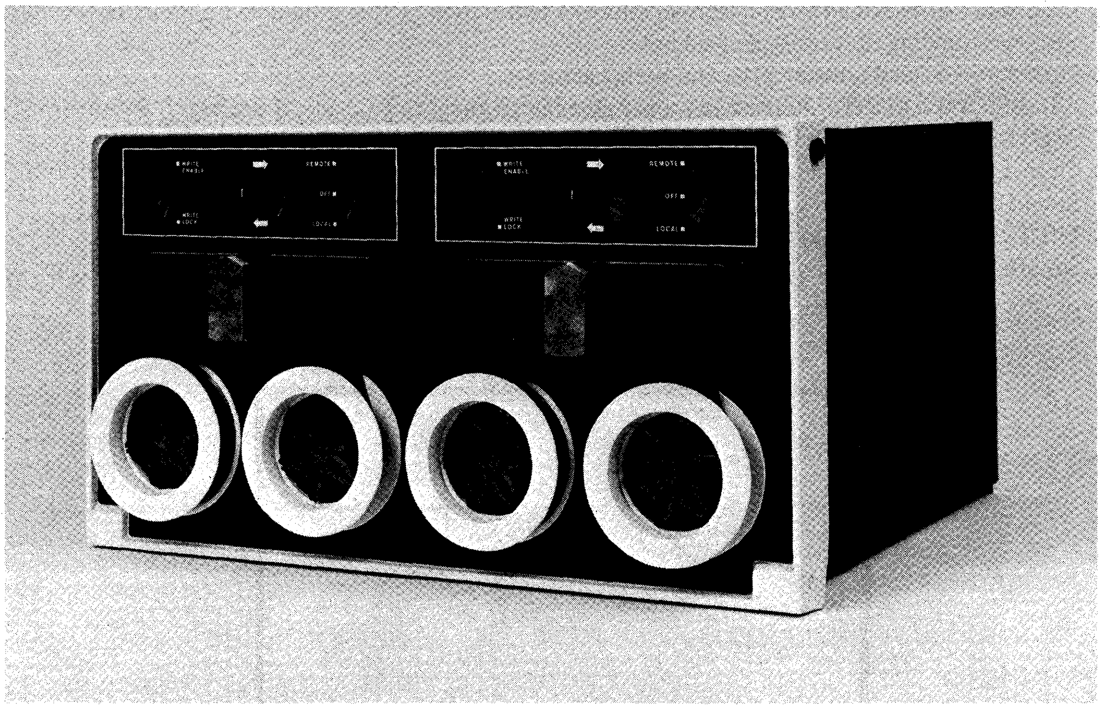


Figure 23-7 TU56 DECtape Transport

DECtape stores and retrieves information at fixed positions on magnetic tape. The advantage of DECtape over conventional magnetic tape is that information at fixed positions can be addressed. Conventional magnetic tape stores information in sequential (not addressable directly), variable-length positions. DECtape incorporates timing and mark information to reference the fixed positions. The ten-channel DECtape records five channels of information; a timing channel, a mark channel, and three information channels. These five channels are duplicated on the five channels remaining to minimize any possibility of information loss from the other channels.

Each formatted (certified) DECtape contains 578 blocks of data consisting of 256 (16-bit) PDP-11 words per block. 562 blocks are available to the user on each DECtape (several blocks are used for file directories).

Tape movement can be controlled by programmed instructions from the computer (i.e., through PIP) or by manual operation of switches located on the front panel of the transport. Data is transferred only under program control. The transport controls and lights are described in Table 23-4.

Operating procedures associated with DECtape units are described below. In order to mount a tape on a DECtape drive:

1. Set the REMOTE/OFF/LOCAL switch to OFF.
2. Place full DECtape reel on left spindle with label facing out. Press reel tightly onto spindle.
3. Pull tape leader over the two tape guides and the magnetic head until it reaches the take-up reel on the right hand side of the tape unit.
4. Wind loose tape end four turns around the empty right-hand reel by rotating right reel clockwise.
5. Set REMOTE/OFF/LOCAL switch to LOCAL. Verify that power is available to the tape unit.
6. Depress FORWARD/REVERSE switch in the FORWARD direction to wind about 15 turns onto the right-hand reel. This ensures that the tape is securely mounted.

**Table 23-4 DECtape Controls**

| Light/Switch | Function |
|---|---|
| REMOTE/OFF/LOCAL | This three-position rocker switch determines control of the DECtape unit.<br><br>REMOTE enables computer control of the transport unit.<br><br>OFF disables the transport unit.<br><br>LOCAL places the transport unit under operator control from the external transport switches. |
| FORWARD/REVERSE | This two-position rocker switch enables manual winding of tape when transport unit is under LOCAL control.<br><br>FORWARD causes tape to feed onto right-hand spool.<br><br>REVERSE causes tape to feed onto left-hand spool. |
| Unit selector | The value specified by this eight-position rotary switch identifies the transport to the computer. A unit selector value of 1 allows the tape on that unit to be accessed as device DT1: |
| WRITE ENABLE/WRITE LOCK | This two-position rocker switch determines whether or not a tape can be written. Any tape can be searched and read; however when the WRITE LOCK switch is on, the tape is protected from accidental writing or program deletion. |
| REMOTE light | When lit, the tape unit is on-line. The tape unit is on-line when:<br><br>1. REMOTE/OFF/LOCAL switch is in REMOTE position, and<br>2. unit selector switch setting agrees with the DECtape unit currently being accessed. |
| WRITE ENABLE light | When lit, indicates that the WRITE ENABLE/WRITE LOCK switch is set to WRITE ENABLE, regardless of whether the REMOTE light is on or not. |

In order to operate the DECtape unit on-line:

1. Set the REMOTE/OFF/LOCAL switch to either LOCAL or OFF and be sure power is available to the system.
2. Load the appropriate DECtape following the instructions above.
3. If the DECtape write operation is to be inhibited (to protect tape from accidental damage), set WRITE ENABLE/WRITE LOCK switch to WRITE LOCK. If the write operation is desired on this tape, set switch to WRITE ENABLE.
4. Dial the correct unit number on the unit selector. (No two active DECtape units should have the same unit number.)
5. Set REMOTE/OFF/LOCAL switch to REMOTE. Use of this DECtape unit is now under program control.

6. When on-line operation of this unit is to cease, set REMOTE/OFF/LOCAL switch to OFF or LOCAL. A moving tape can be stopped by quickly switching the REMOTE/OFF/LOCAL switch from REMOTE to LOCAL. The switch can be set to OFF when tape motion has stopped.

To remove a DECtape from the tape unit:

1. Set REMOTE/OFF/LOCAL switch to LOCAL.
2. Depress and hold REVERSE switch until all tape is wound onto the left-hand tape reel.
3. Set REMOTE/OFF/LOCAL switch to OFF.
4. Remove full reel from left-hand spindle.

## 23.6 TM11/TU10 AND TJU16 MAGTAPE CONTROL AND TRANSPORT

Magtape is an optional addition to RSTS/E system. Magtape is used to provide storage for large volumes of data and programs. Writing, reading, and search operations are performed in a serial manner. Transfer of information can be made between RSTS/E and other computer systems because TJU16 and TM11 Controllers read and write information in industry-compatible format.

The basic DECmagtape system consists of:

1. The TU10 Transport, pictured in Figure 23-8, can read or write information on magnetic tape in seven or nine channels (tracks). The TJU16 Tape Transport reads and writes information on nine channels only. The TU10 reads and writes data at 800 bits per inch (BPI), and the TJU16 Tape Transport can be used to read and write magtape either at 800 BPI or at 800 BPI and 1600 BPI.
2. The TM11 Controller is an interface between the TU10 Tape Transport and the PDP-11 system. The RM11 or RH70 is an interface between the PDP-11 and the TM02 Formatter/TU16 Transport. One controller, transparent to the RSTS/E user, serves up to eight transport units.

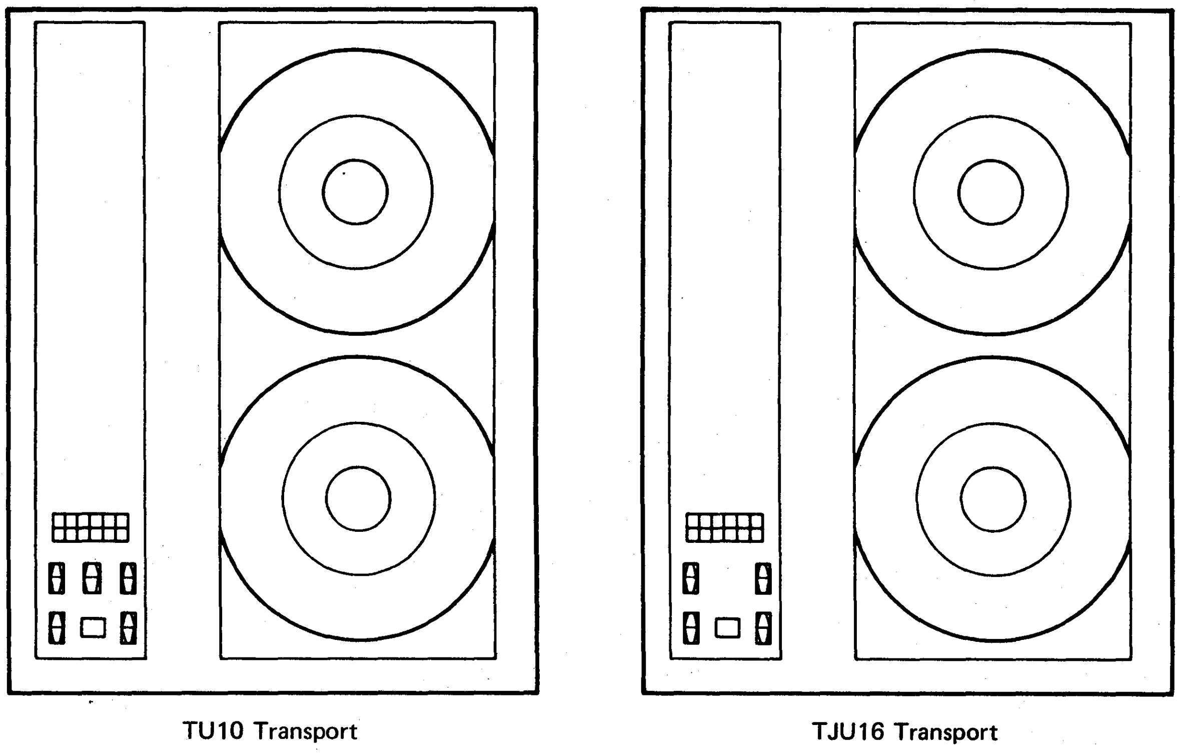
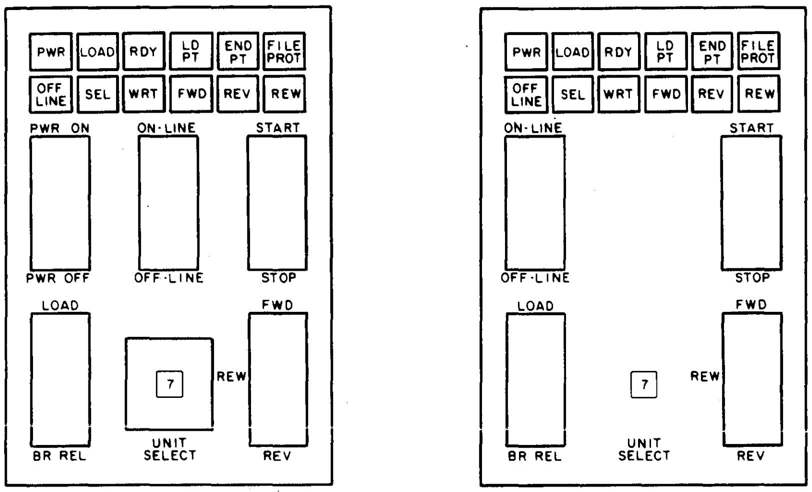


Figure 23-8 DEC Magnetic Tape System

Transfer rate is up to 36,000 characters per second for the TU10 and up to 72,000 characters per second for the TJU16. Ten and one half inch magtape reels permit up to 2400 feet of tape per reel. Rewind time for a reel of 2400 feet is approximately 3 minutes, end to end.

## 23.6.1 Magtape Control Panel

The magtape transport control panel is shown in Figure 23-9. This panel is located at the lower left of the TU10 and TJU16 front panel shown in Figure 23-8. Table 23-5 describes the tape transport controls and Table 23-6 describes the various tape transport indicators.



TU10 Control Panel    TJU16 Control Panel

Figure 23-9 Control Panels

## 23.6.2 Magtape Operating Procedures

Whenever handling magnetic tapes and reels, it is important to observe the following precautions to prevent loss of data and/or damage to tape handling equipment:

1. Handle a tape reel by the hub hole only. Squeezing reel flanges can damage tape edges when winding or unwinding tape.
2. Never touch tape between BOT and EOT markers. Do not allow end of tape to drag on floor.
3. Never use a contaminated reel of tape; this spreads dirt to clean tape reels and can affect transport operation.
4. Always store tape reels inside containers. Keep empty containers closed so dust and dirt cannot collect.
5. Inspect tapes, reels, and containers for dust and dirt. Replace old or damaged take-up reels.
6. Do not smoke near transport or tape storage area. Smoke and ash are especially damaging to tape.
7. Do not place transport near a line printer or other device that produces paper dust.
8. Clean tape path frequently.

To mount a tape reel on the magtape transport.

1. Apply power to the transport. (Depress the PWR ON switch on the TU10 transport.) Ensure that the LOAD/BR REL switch is in the center position and that the ON-LINE/OFF-LINE switch is in its OFF-LINE position.
2. Place a write-enable ring in the groove on the file reel if data is to be written on the tape. If writing is not required, be sure there is no ring in the groove.
3. Mount file reel onto lower hub with groove facing toward the back. Press reel tightly onto spindle; tighten center nut.

(continued on next page)

**Table 23-5 Magtape Transport Controls**

| Switch | Function |
|---|---|
| PWR ON/PWR OFF | This two-position switch applies power to the TU10 Transport. (This switch does not exist on the TJU16 Transport.) |
| ON-LINE/OFF-LINE | This two-position switch controls operation of the transport unit. ON-LINE allows system operation under program control; OFF-LINE allows manual operation. Tape unit cannot be remotely selected when switch is in OFF-LINE position. |
| START/STOP | This two-position switch controls starting and stopping of tape motion. STOP does not stop transport during a rewind operation. |
| LOAD/BR REL | This three-position switch energizes the vacuum system in the LOAD position (necessary for any operation). The BR REL position releases vacuum tension and allows reels to be manually rotated..Center position locks reel brakes. |
| UNIT SELECT | This eight-position rotary switch identifies the transport to the computer. A unit select value of 1 allows the tape on that unit to be accessed as device MT1:. No two transports should be set to the same number. |
| FWD/REW/REV | This three-position switch moves the tape in the selected direction, depending on activation of the START/STOP switch. FWD moves tape forward until BOT or EOT marker is sensed. REW rewinds tape onto the feed reel until BOT marker is sensed. REV rewinds tape until BOT marker is sensed; toggling the START/STOP switch again causes the tape to rewind off the reel. |

4. Install take-up reel (top reel), if necessary, as described in (3) above. The top reel is generally permanent and should not require installation by the user.
5. Place LOAD/BR REL switch to the BR REL position.
6. Unwind tape from the file reel and thread tape over tape guides and head assembly as shown in Figure 23-10. Wind about five turns of tape onto take-up reel.
7. Set LOAD/BR REL switch to LOAD position to draw tape into vacuum columns. As a result, the LOAD light comes on.
8. Select FWD and depress the START switch to advance the tape to the load point. When BOT marker is sensed, tape motion stops, the FWD indicator goes out and the LOAD PT indicator comes on.

   If tape motion continues for more than 10 seconds, depress STOP, select REV, and then depress START. The tape will advance to the BOT marker before stopping. (This may be necessary if, in winding the tape manually, the BOT marker has already been passed.)

Setting the ON-LINE/OFF-LINE switch to ON-LINE allows the transport to accept commands from the controller under program control. The transport is not fully on-line until the RDY and SEL indicators are lit.

To remove a tape from the transport unit:

1. Set ON-LINE/OFF-LINE switch to OFF/LINE position.
2. Set START/STOP switch to STOP position.
3. Set FWD/REW/REV switch to REW position.

Table 23-6 Magtape Transport Indicators

| Light | Function |
|---|---|
| PWR light (power) | When lit, indicates that power is available to the transport unit. |
| LOAD light | When lit, indicates that vacuum system has been enabled, allowing either on-line or off-line commands. |
| RDY light (ready) | When lit, indicates that all I/O lines are enabled. Transport can accept processor commands provided SEL light is also lit. |
| LD PT light (load point) | When lit, indicates that BOT marker has been sensed; transport ready for operation. |
| END PT light (end point) | When lit, indicates that EOT marker has been sensed; all tape motion stops to prevent tape from winding off reel. |
| FILE PROT light (file protection) | When lit, indicates that writing on the tape is inhibited. This is true if no file reel is mounted on feed reel hub or if a file reel is mounted without a write enable ring. |
| OFF-LINE light | When lit, indicates that transport can be operated manually and cannot be operated under program control. |
| SEL light (select) | When lit, indicates that transport has been selected and is completely on-line. Transport can read or write data. |
| WRT light (write) | When lit, indicates that write-enable ring has been installed on feed reel and transport can write on tape. |
| FWD light (forward) | When lit, indicates tape is moving in forward direction. |
| REV light (reverse) | When lit, indicates that tape is moving in reverse direction. |
| REW light (rewind) | When lit, indicates that tape is being rewound; Tape continues until BOT marker is sensed. |

4. Set START/STOP switch to START position. Tape rewinds until BOT marker is reached.
5. Set LOAD/BR REL switch to BR REL position to release brakes.
6. Gently hand wind the file reel in a counter clockwise direction until all of the tape is wound onto the reel. Do not jerk the reel. This may stretch or compress the tape which can damage data.
7. Remove the file reel from the hub assembly.

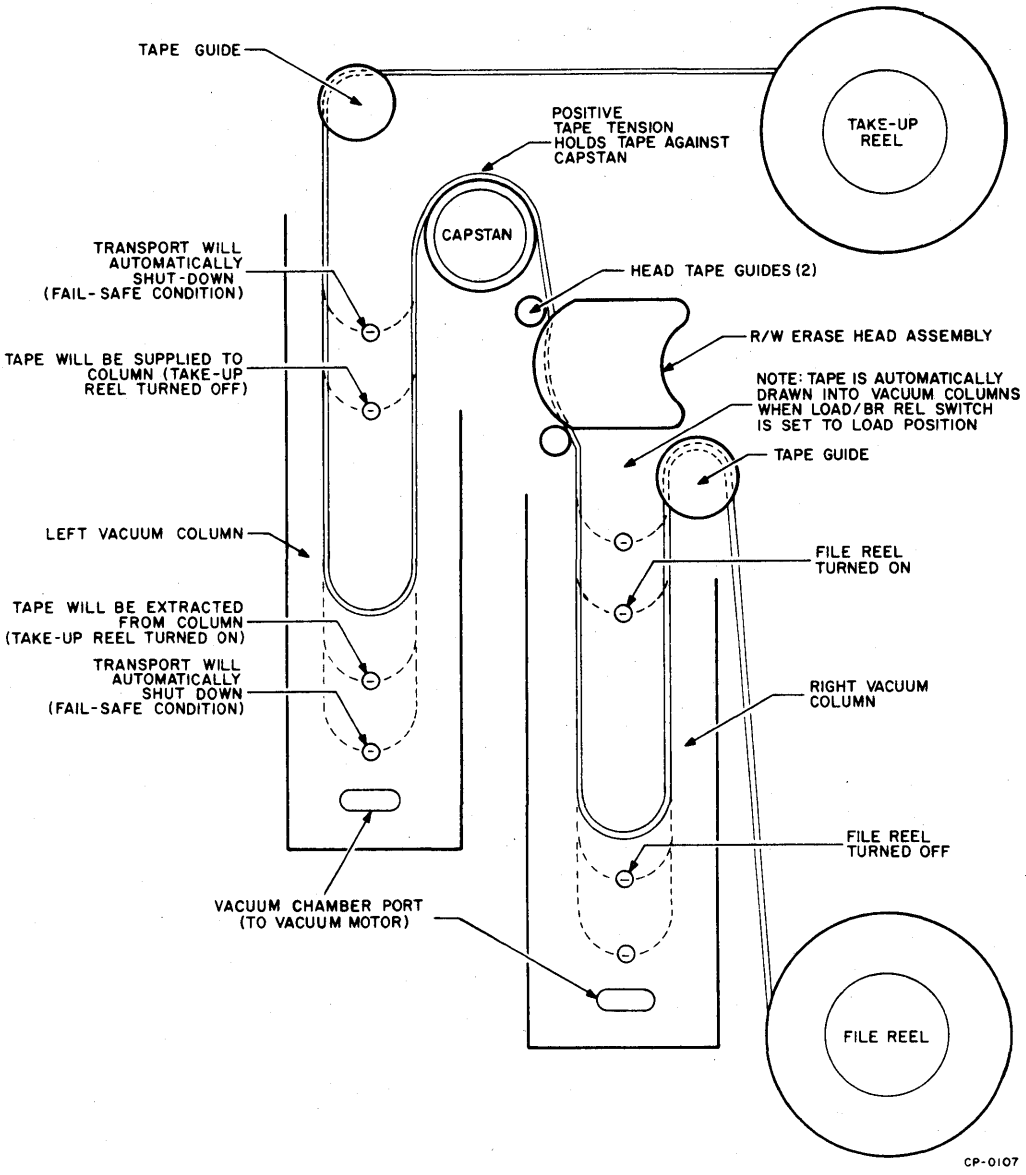

TAPE GUIDE
POSITIVE TAPE TENSION HOLDS TAPE AGAINST CAPSTAN
TAKE-UP REEL
CAPSTAN
TRANSPORT WILL AUTOMATICALLY SHUT-DOWN (FAIL-SAFE CONDITION)
HEAD TAPE GUIDES (2)
R/W ERASE HEAD ASSEMBLY
TAPE WILL BE SUPPLIED TO COLUMN (TAKE-UP REEL TURNED OFF)
NOTE: TAPE IS AUTOMATICALLY DRAWN INTO VACUUM COLUMNS WHEN LOAD/BR REL SWITCH IS SET TO LOAD POSITION
TAPE GUIDE
LEFT VACUUM COLUMN
FILE REEL TURNED ON
TAPE WILL BE EXTRACTED FROM COLUMN (TAKE-UP REEL TURNED ON)
TRANSPORT WILL AUTOMATICALLY SHUT DOWN (FAIL-SAFE CONDITION)
RIGHT VACUUM COLUMN
FILE REEL TURNED OFF
VACUUM CHAMBER PORT (TO VACUUM MOTOR)
FILE REEL
CP-0107

Figure 23-10 Magtape Transport Threading Diagram

## 23.7 VT05 ALPHANUMERIC DISPLAY TERMINAL

The VT05 alphanumeric display terminal consists of a cathode ray tube (CRT) display and a self-contained keyboard. The VT05 keyboard operates as a typewriter keyboard except that no hard copy is produced. Each graphic character generated by typing at the keyboard is converted to a 5 by 7 dot matrix and displayed on a television-like screen. The full capacity of the screen is 20 lines, each containing 72 character positions for a total of 1440 characters.

When power is applied to the terminal and the CRT filament is warmed up, a blinking indicator called the cursor appears at the leftmost position of the top line (line number 1) of the screen. The blinking cursor indicates the position which the next generated character will occupy on the screen. The cursor can be moved up, down, left, or right by the use of various control characters generated by keys located to the right of the keyboard. When a displayable character is generated, its representation is displayed and the cursor automatically moves right to the next character location until the cursor reaches character position 72.

A speaker in the VT05 emits an audible tone or beep when the cursor reaches character position 65. This action warns the user that the cursor is within 8 spaces of the end of the line. (The speaker also beeps when the terminal or the computer generates the BEL character.) When the cursor reaches position 72 on a line, characters subsequently generated replace the character previously in position 72. Automatic carriage return or line feed is a hardware modification and terminals without the modification must be programmed to include these automatic operations.

The VT05 keyboard, shown in Figure 23-11, is similar to a typewriter keyboard except for the following operations keys:

| | |
|---|---|
| ALT | Generates the ESC character CHR$(27). |
| CTRL | Control key is used to generate various control character combinations. See Chapter 3 for a description of control characters. |
| LF | Generates the LINE FEED character CHR$(10). |
| CR | Generates the RETURN character CHR$(13). |
| RUBOUT | Generates the DEL character CHR$(127). |
| TAB | Generates the HT character CHR$(9) and causes the cursor to move to the right to the next tab stop. Tab stops are preset eight character spaces apart and are at locations 1, 9, 17, 25, 33, 41, 49, 57 and 65. Once the cursor reaches character position 65, the HT character moves the cursor right one position. Another HT character causes a RETURN and LINE FEED, leaving the cursor at position 1. |

A line feed typed with the cursor in the bottom line (line 20) causes all displayed data on the screen to move up one line and any data in the top line to disappear. This action is termed automatic scrolling and is useful when a large amount of data is transmitted from the computer to the VT05 and is to be received and displayed. The user can employ the CTRL/S combination described in Chapter 5 to temporarily suspend transmission of such output to the screen and enable examination of data currently displayed on the screen. The CTRL/Q combination resumes output.

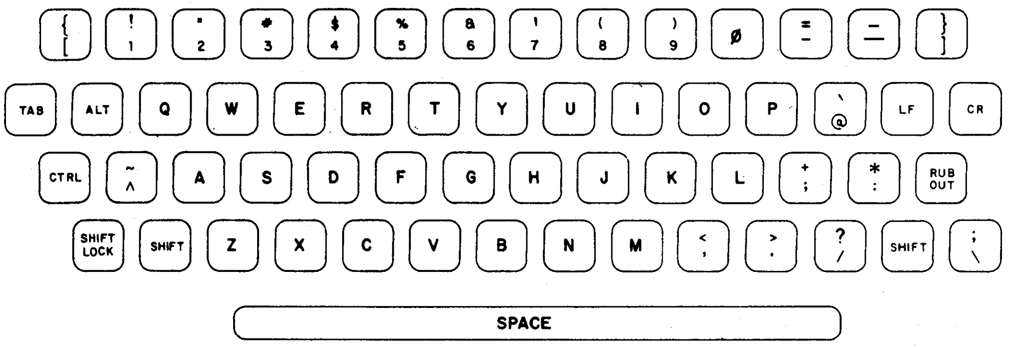


Figure 23-11 VT05 Keyboard

Four actions erase characters from the screen. A character is erased when the cursor is placed under it and a space character is generated. A character is replaced on the screen if the cursor is positioned under an existing character and another character is generated. The EOL and EOS special functions also erase characters from the screen.

The DEL code, CHR$(127), generated by typing the RUBOUT key, is ignored and no visual indication occurs on the display. When the RUBOUT key is typed on a VT05 terminal directly connected to a RSTS/E system, the system backspaces the cursor one character position, generates a space character in that position on the screen and in memory and backspaces one character position again. A RUBOUT key typed on a VT05 terminal connected to a RSTS/E system by a dial up line is treated as the RUBOUT key typed at a teleprinter unless the TTYSET SCOPE command is in effect for that line. Executing the SCOPE command causes the RUBOUT key to be treated as described above.

The ALT key typed at a VT05 generates the ESC character. When received by the VT05, the ESC character has no effect on the display. RSTS/E system programs such as GRIPE treat the ESC character as a line terminating character; the system echoes the $ character on the display when the ESC character is received.

Controls to adjust the quality of the display are located on the right hand side of the terminal shown in Figure 23-12. Means to select a baud rate and mode of operation are on the rear panel of the device. At speeds above 300 baud, FILL characters for time delay as shown in Table 23-7 are required after some control characters are generated. The number of FILL characters required depends upon the baud rate at which the terminal operates.

**Table 23-7 FILL Characters Required for VT05**

| Baud Rate | Number of FILL Characters |
|---|---|
| 300 | none |
| 600 | 1 |
| 1200 | 2 |
| 2400 | 4 |

To effect the proper number of FILL characters on a VT05 terminal which operates above 300 baud and whose characteristics are not permanently set, use the FILL command of the TTYSET system program.

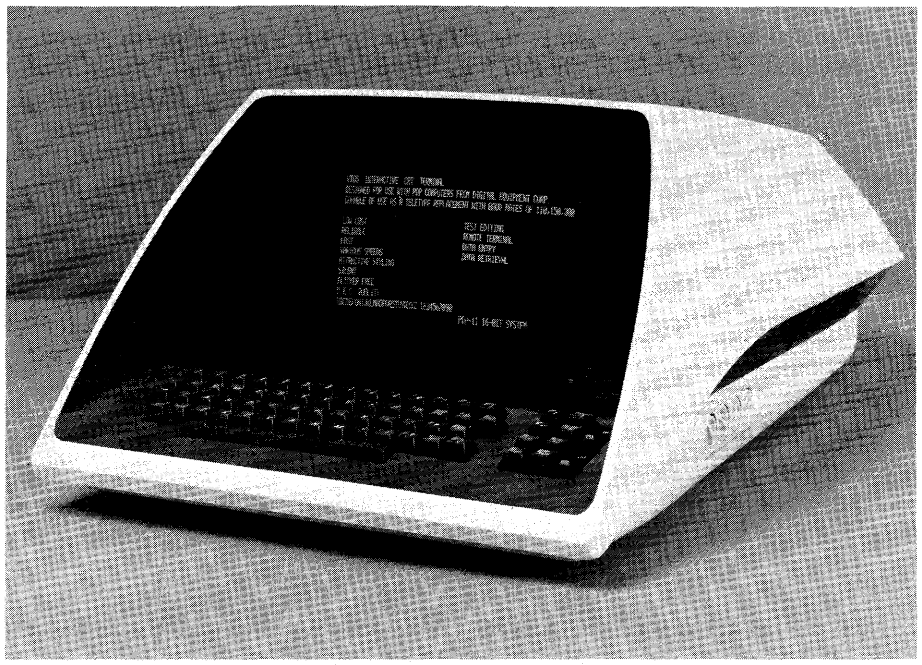


Figure 23-12 VT05 Alphanumeric Display Terminal

### 23.7.1 Controls and Operating Procedures

The controls and switches for a VT05 terminal are listed and described in Table 23-8. To start the terminal connected by a direct line to the RSTS/E system, do the following.

1. Set the LOC/REM switch to the REM position.
2. Set the ON/OFF switch to the ON position. Allow approximately one minute for the filament to warm up, for the blinking cursor to appear in the home position (upper left corner of the display), and for the speaker to emit one beep.
3. If the cursor fails to appear as specified, press the HOME key. Ensure that the BRIGHTNESS control is not turned fully counterclockwise. If the cursor still fails to appear, place the ON/OFF switch in its OFF position and report the malfunction to the proper person.
4. To properly adjust the clarity of the display, turn the CONTRAST control counterclockwise to its minimum setting. Turn the BRIGHTNESS control counterclockwise until the characters are barely visible. Adjust the CONTRAST control to the optimum level.
5. When the operating session is completed, set the ON/OFF switch to its OFF position.

To operate the VT05 terminal which is connected to the computer by a dial up line, perform the following steps.

1. Follow the procedures to start the VT05 terminal as if it were connected by a direct line to the RSTS/E system and set the BAUD RATE selector switch on the rear panel to its 110 position or to the position required by the permanent default characteristics established by the system manager for that line.
2. Dial the RSTS/E system and perform the log in procedures. If the line does not have permanent default characteristics for a VT05, continue with step 3. Otherwise, go to step 5.
3. Run the TTYSET system program and type the VT05 macro command or the SPEED command with the proper baud rate value. (The maximum baud rate allowed on a voice grade telephone line is 300.)
4. Set the BAUD RATE selector switch on the rear panel to the proper value, after which the VT05 operates at the proper baud rate.
5. When the operating session is completed, set the BAUD RATE selector switch to the 110 position and set the ON/OFF switch to its OFF position.

## 23.8 2741 COMMUNICATIONS TERMINALS

RSTS/E systems allow the use of 2741 compatible[1] terminals for time sharing operations. 2741 terminals employ a Selectric typing mechanism which provides high quality copy in upper and lower case format. Such copy is expecially suitable for documentation preparation and for output of the RUNOFF system program.

The 2741 terminal operates in half duplex mode and transmits and receives non-ASCII characters. The terminal echoes locally; the computer does not echo the characters. This action differs greatly from ASCII terminals which operate under RSTS/E in full duplex mode.

Half duplex operation of the 2741 terminal involves stricter intercommunication between the device and the computer. For example, when the user types the RETURN key to enter a line, the keyboard locks until the computer accepts the line and sends an EOT character to unlock the keyboard. This locking and unlocking action is noticeable and may be annoying to the fast typist.

Many graphic code and keyboard arrangements are available for 2741 compatible terminals. RSTS/E supports the four most common codes: Correspondence, Extended Binary Coded Decimal, Binary Coded Decimal and CALL 360 BASIC. Since the system can be configured for any combination of the four codes, it is advisable to consult the system manager for information concerning which codes are available at the local installation.

2741 terminals do not generate all the characters normally used for time sharing operations under RSTS/E. The system interprets certain keys in special ways described in the following subsections. The four supported keyboard arrangements are shown in Figures 23-13 through 23-16 at the end of the section.

[1]The RSTS/E 2741 code has been tested with IBM, DATEL, and TREND DATA terminals. Digital Equipment Corporation makes no commitment to support 2741-type terminals made by other manufacturers.

**Table 23-8 VT05 Controls and Switches**

| Location | Label | Operation |
|---|---|---|
| Keyboard | ON/OFF | When placed in its ON position, power is applied to the VT05 display and the refresh memory is cleared. When placed in its OFF position, the VT05 is inoperative and the screen is darkened. |
| | LOC/REM | When placed in its LOC position, it breaks the electrical connection between the device and the computer. Local operation for maintenance and training is still allowed. In its REM position, it connects the terminal to the remote computer system. Data is simultaneously transmitted to and received from the computer. |
| Right Side | BRIGHTNESS | Turning this control in the clockwise direction increases the brightness of the screen. Turning the control counterclockwise decreases brightness. If turned fully counterclockwise, the display is completely darkened. |
| | CONTRAST | This control increases and decreases the clarity of the characters displayed on the screen. |
| | VERTICAL | This control synchronizes the display in the vertical position such that all 20 lines are visible on the screen. |
| | HORIZONTAL | This control moves the display in the horizontal direction such that all 72 character positions are visible on the screen. |
| Rear Panel | BAUD RATE | This 10-position selection switch determines the rates at which the terminal transmits and receives data. |
| | FULL/HALF DUPLEX | When at FULL, it allows the keyboard to transmit data to the computer and allows the display to simultaneously receive data from the computer. When at HALF, data is transmitted to the VT05 receiver logic as well as to the computer. |

### 23.8.1 The ATTN Key

The 2741 terminal has an ATTN (Attention) key usually on the upper right hand portion of the keyboard. (Some terminals have an equivalent key called the BREAK key.) The key has several uses under RSTS/E depending upon whether the terminal is transmitting data to or receiving data from the computer.

The terminal is considered transmitting data to the computer when it is at BASIC-PLUS command level or in the program input state. The terminal is receiving characters whenever the system performs output to the device.

When the terminal is transmitting data, the ATTN key can have two effects. If pressed while the SHIFT key is in its upper case position, the key generates the effect of a CTRL/C combination as described in Chapter 5. The system echoes the transmission as either ^C or ↑C. If pressed while the SHIFT key is in its lower case position, the key generates the effect of a CTRL/U combination (erase line) as described in Chapter 5. The system echoes the lower case ATTN key as either ^U or ↑U.

When the terminal is receiving data, pressing the ATTN key with the SHIFT key in its upper case or lower case position generates the effect of a CTRL/C combination as described in Chapter 5. This action allows the user to interrupt computer printout and return control to BASIC-PLUS command level.

**NOTE**

The system reacts to the CTRL/C combination on a 2741 type terminal as if the user typed two such combinations in rapid succession. Programs executing the CTRL/C trap enable system function cannot trap a CTRL/C combination from a 2741 type terminal.

### 23.8.2 The RETURN Key

The RETURN key generates one of two characters depending upon whether the SHIFT key is in its upper or lower case position. If typed as a lower case character, the key generates a CR character (CHR$(13)). If typed as an upper case character, the key generates a LF character (CHR$(10)). The system thus allows the user to continue BASIC-PLUS statement lines when he enters source code from a 2741 terminal.

### 23.8.3 The BKSP Key

The BKSP key generates one of two characters depending upon whether the SHIFT key is in its upper or lower case position. If typed as an upper case character, the key backspaces the typing element one character position and generates the BACKSPACE character (CHR$(8)). If typed as a lower case character, the key backspaces the typing element one character position and generates the DEL (RUBOUT) character (CHR$(127)). In the latter case, the system deletes from memory the last character typed. This action is similar to the RUBOUT key typed at an ASCII terminal except that any replacement character typed is printed over the deleted character.

### 23.8.4 Bracket Characters

Since most 2741 terminals have no bracket characters, the system accepts ( and ) characters typed in place of [ and ] characters. This feature allows users to delimit a project-programmer number with open and close parenthesis characters rather than with bracket characters. For example, (100,100) is equivalent to [100,100]. System programs translate [ and ] characters to ( and ) characters before processing commands.

### 23.8.5 Changing Codes

If the system is configured to handle more than one transmission code, the user can change the code recognized by the system. This feature is available because the system initializes each device to a default code when time sharing operations begin. Therefore, if an individual terminal does not employ the default code, the user must change the code to operate on the system.

To enable the system to recognize codes generated by one of the keyboards shown in Figure 23-13 through 23-16, the user must type the numeral 1 followed by the upper case ATTN key. (See the description of the ATTN key in Section 23.8.1.) For example,

```
1↑C

E963

Ready
```

As a result, the system changes the code conversion table for the device. It prints an identifier which associates both the code and its required typing sphere and prints the Ready message. Table 23-9 shows the identifiers and related reference information.

**Table 23-9 2741 Transmission Code Identifiers**

| Identifier | Keyboard Code | Figure | Typing Sphere |
|---|---|---|---|
| C029 | Correspondence | 23-13 | 1167-029 |
| E963 | EBCD | 23-14 | 1167-963 |
| B938 | BCD | 23-15 | 1167-938 |
| C087 | CALL/360 BASIC | 23-16 | 1167-087 |

The code table the system uses and the typing sphere the device uses must match the particular keyboard. If either the typing sphere or the code table is incorrect, the terminal produces garbled output. For example,

```
1'g

X92#

Nupvi
```

To verify the typing sphere, the user can compare the typing sphere number shown in Table 23-9 with that stamped on the top edge of the type ball under the retaining clamp lever. To verify the keyboard, compare it with the one shown in the figure referred to in Table 23-9. If the typing sphere does not match the keyboard, install the correct type ball. Otherwise, continue changing codes until the system prints a recognizable identifier and Ready message. If the user changes codes four times and the system does not print a recognizable message, the terminal cannot be used unless the system is regenerated to support that code and keyboard arrangement.

The graphics of the typing spheres for each keyboard are compatible with conventional system graphics except for certain special characters. Where differences exist, the conventional character is shown in the related keyboard layout above the graphic printed by the sphere. All system programs accept lower case characters by performing a conversion to upper case with the CVT$$ function described in Section 12.5 of the *BASIC-PLUS Language Manual.*

**Correspondence Code Keyboard**

The following special characters and system actions are produced on this keyboard.

| Character or Action | 2741 Method |
|---|---|
| LF (LINE FEED) | RETURN key in upper case |
| CR (RETURN) | RETURN key in lower case |
| DEL (RUBOUT) | BKSP key in lower case |
| BKSP (BACKSPACE) | BKSP key in upper case |
| CTRL/C combination | ATTN key in upper case |
| CTRL/U combination | ATTN key in lower case |
| CTRL/O combination | None |
| CTRL/Z combination | None |
| Change Code | 1 followed by the upper case ATTN key |

Figure 23-13 shows the correspondence keyboard layout and specifies the required typing sphere. Special system characters such as ^, and \, which do not have a graphic equivalent are shown in the area above the key that is used as a replacement. For example, the terminal prints the ↑ character to designate the ^ character. Abbreviations above or below keys are defined in the legend.

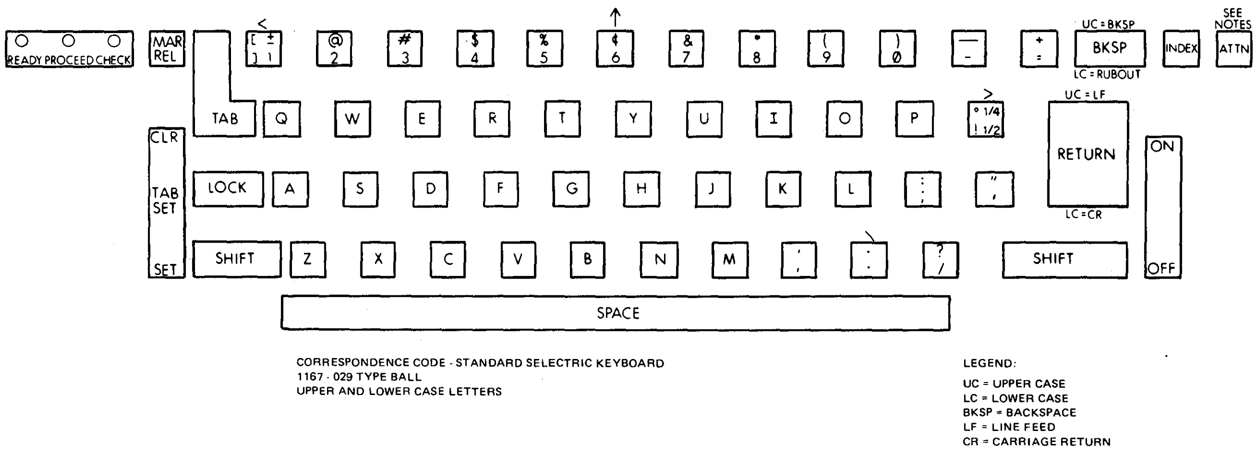


Figure 23-13 Correspondence Code Keyboard

**EBCD Keyboard**

The following special characters and system actions are produced on this keyboard.

| Character or Action | 2741 Method |
|---|---|
| LF (LINE FEED) | RETURN key in upper case |
| CR (RETURN) | RETURN key in lower case |
| DEL (RUBOUT) | BKSP key in lower case |
| BKSP (BACKSPACE) | BKSP key in upper case |
| CTRL/C combination | ATTN key in upper case |
| CTRL/U combination | ATTN key in lower case |
| CTRL/O combination | None |
| CTRL/Z combination | None |
| Change Code | 1 followed by the upper case ATTN key |

Figure 23-14 shows the EBCD keyboard layout and specifies the required typing sphere. Special system characters such as ^, and \ which do not have a graphic equivalent are shown in the area above the key that is used as a replacement. For example, the terminal prints a ↑ character in place of the ^ character. Abbreviations above or below keys are defined in the legend.

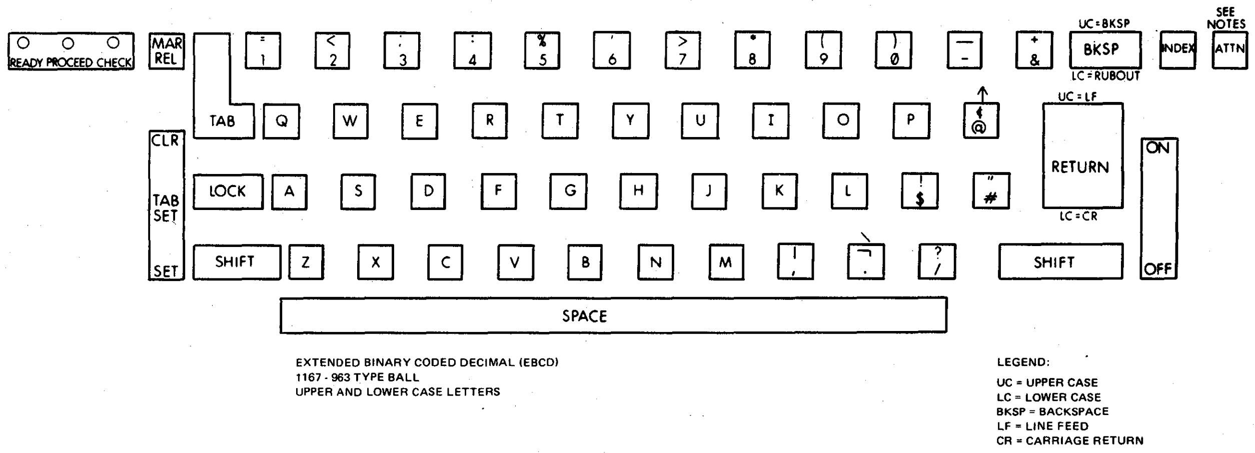


Figure 23-14 EBCD Keyboard

## BCD Keyboard

The following special characters and system actions are produced on this keyboard.

| Character Action | 2741 Method |
|---|---|
| LF (LINE FEED) | RETURN key in upper case |
| CR (RETURN) | RETURN key in lower case |
| DEL (RUBOUT) | BKSP key in lower case |
| BKSP (BACKSPACE) | BKSP key in upper case |
| CTRL/C combination | ATTN key in upper case |
| CTRL/U combination | ATTN key in lower case |
| CTRL/O combination | None |
| CTRL/Z combination | None |
| Change Code | 1 followed by the upper case ATTN key |

Figure 23-15 shows the BCD keyboard layout, and specifies the required typing sphere. Special system characters such as ^, and \, which do not have a graphic equivalent are shown in the area above the key that is used as a replacement. For example, the terminal prints the ↑ character to designate the ^ character. Abbreviations above or below keys are defined in the legend.

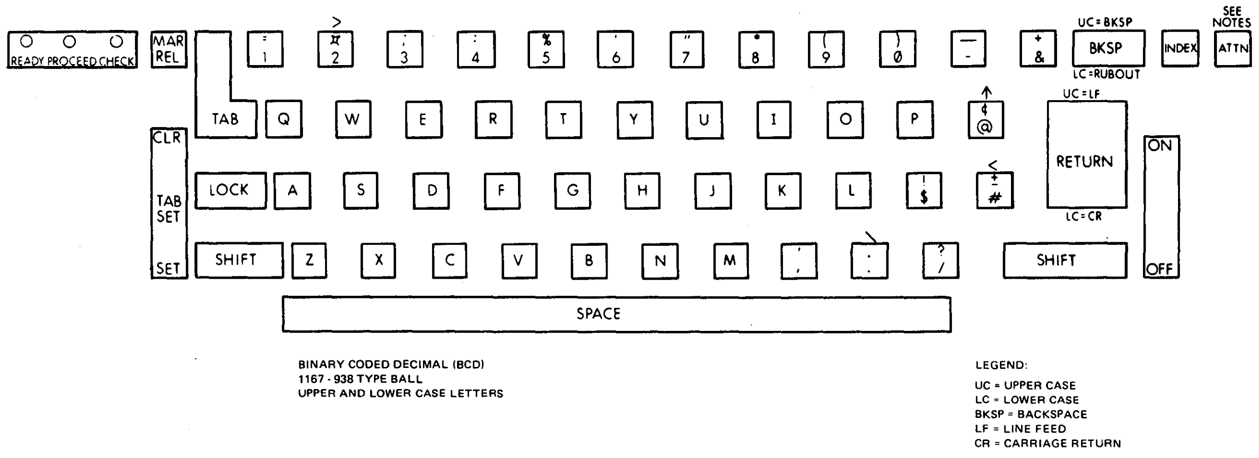


Figure 23-15 BCD Keyboard

## CALL/360 BASIC Keyboard

The following special characters and system action are produced on this keyboard.

| Character or Action | 2741 Method |
|---|---|
| LF (LINE FEED) | RETURN key in upper case |
| CR (RETURN) | RETURN key in lower case |
| DEL (RUBOUT) | BKSP key in lower case |
| BKSP (BACKSPACE) | BKSP key in upper case |
| CTRL/C combination | ATTN key in upper case |
| CTRL/U combination | ATTN key in lower case |
| CTRL/O combination | None |
| CTRL/Z combination | None |
| Change Code | 1 followed by the upper case ATTN key |

Figure 23-16 shows the CALL/360 BASIC keyboard layout and specifies the required typing sphere. Special system characters such as ^, \, [, ?, and ] which do not have a graphic equivalent are shown in the area above the key that is used as a replacement. For example, the terminal prints a ↑ character to designate the ^ character. Abbreviations above or below keys are defined in the legend. This keyboard does not produce lower case letters.

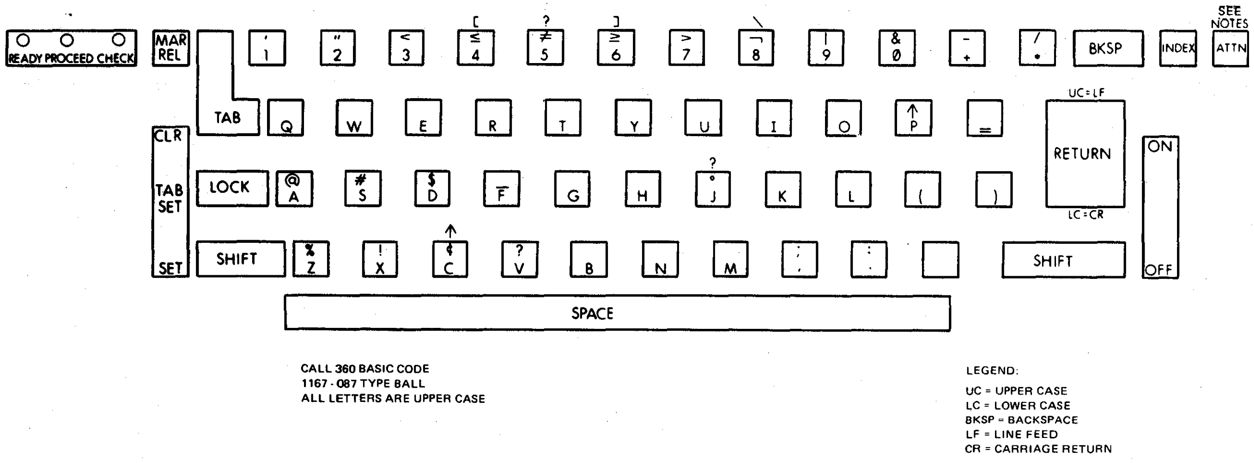


Figure 23-16 CALL/360 BASIC

## 23.9 LA36 DECWRITER II OPERATOR CONTROLS

DECwriter II offers fast, reliable operation and can be easily interfaced as a remote terminal or local computer I/O device. This terminal prints 30 characters per second and up to 132 characters per line. Characters are formed from a 7 x 7 dot matrix. Character spacing is 10 characters per inch, horizontal and 6 lines per inch, vertical. An original and up to five copies can be printed, and forms can be any width from 3 inches to 15 inches wide.

Table 23-10 describes the purpose of each operator control on the DECwriter II.

**Table 23-10 DECwriter II Operator Controls**

| Control | Meaning |
|---|---|
| Power On-Off | Applies and removes AC power to entire machine. |
| Line/Local | Selects on-line or local operation. |
| Baud-rate – 110, 150, 300 | 3-position switch selects the baud rate clock frequency for communications line operation. |
| Forms thickness adjustment | Located on right side of print head carriage. Selects proper gap for 1- through 6-part form. Approximately 1 click for each part. |
| Right Tractor Adjustment | Thumb screw may be loosened to allow movement of right tractor for various forms widths. |
| Fine Vertical Tractor Release | Line feed knob may be depressed inward and rotated in the approximate direction for precise location of printing with respect to vertical zones. |

## 23.10 RX11 FLOPPY DISK

The RX11 floppy disk system provides a low cost, random access, mass memory device capable of storing up to 256,256 8-bit bytes of data in a non-file structured format. The floppy disk itself is a thin flexible, oxide-coated disk, similar in size to a 45 RPM phonograph record. The disk is recorded on one side and is housed in an 8 inch square flexible envelope. The envelope has a large center hole for the drive spindle, a small hole for track index sensing, and a larger slit for the read/write head.

Each RX11 floppy disk controller handles one or two drives, mounted side-by-side horizontally. The lower numbered unit is on the left; the higher numbered unit on the right. RSTS/E systems support up to four controllers (and eight drives).

Once power is applied to the floppy disk system, raise the door of the desired disk drive by pulling up on the centrally located latch. Then simply insert the floppy disk (still housed in its square envelope) into the drive, label-side up, and read/write head slit first. Close the door firmly. The floppy disk is now ready to use; the disk rotates at its operating speed immediately.

To remove the disk, simply open the door and pull the disk envelope out. Once again, the disk stops rotating as soon as the door is opened.

# APPENDIX A
# BASIC-PLUS LANGUAGE SUMMARY

All manual section numbers given in this Appendix refer to sections in the *BASIC-PLUS Language Manual.*

## A.1 SUMMARY OF VARIABLE TYPES

| Type | Variable Name | Examples | |
|---|---|---|---|
| Floating Point | single letter optionally followed by a single digit | A<br>I<br>X3 | |
| Integer | any floating point variable name followed by a % character | B%<br>D7% | |
| Character String | any floating point variable name followed by a $ character | M$<br>R1$ | |
| Floating Point Matrix | any floating point variable name followed by one or two dimension elements in parentheses | S(4)<br>N2(8) | E(5,1)<br>V8(3,3) |
| Integer Matrix | any integer variable name followed by one or two dimension elements in parentheses | A%(2)<br>E3%(4) | I%(3,5)<br>R2%(2,1) |
| Character String Matrix | any character string variable name followed by one or two dimension elements in parentheses | C$(1)<br>A2$(8) | S$(8,5)<br>V1$(4,2) |

## A.2 SUMMARY OF OPERATORS

| Type | Operator | | Operates Upon |
|---|---|---|---|
| Arithmetic | - | unary minus | numeric variables and constants |
| | ↑ | exponentiation | |
| | *,/ | multiplication, division | |
| | +,- | addition, subtraction | |
| Relational | = | equals | string or numeric variables and constants |
| | < | less than | |
| | <= | less than or equal to | |
| | > | greater than | |
| | >= | greater than or equal to | |
| | <> | not equal to | |
| | == | approximately equal to | numeric variables |
| | == | exactly equal to | string variables |
| Logical | NOT | logical negation | integer variables and integer-valued expressions |
| | AND | logical product | |
| | OR | logical sum | |

| Type | Operator | | Operates Upon |
|---|---|---|---|
| Logical (Cont.) | XOR | logical exclusive or | |
| | IMP | logical implication | |
| | EQV | equivalence | |
| String | + | concatenation | string constants and variables |
| Matrix | +,- | addition and subtraction of matrices or equal dimensions, one operator per statement | dimensioned variables. See Section 7.6.1 for further details. |
| | * | multiplication of conformable matrices | |
| | * | scalar multiplication of a matrix, see Section 7.5.1. | |

## A.3 SUMMARY OF FUNCTIONS

Under the Function column, the function is shown as:

Y=function

where the characters % and $ are appended to Y if the value returned is an integer or character string.

A floating value (X), where specified, can always be replaced by an integer value. An integer value (N%) can always be replaced by a floating value (an implied FIX is done) except in the CVT%$ and MAGTAPE functions (the symbol I% is used to indicate the necessity for an integer value).

Section numbers found in the Explanation column refer to sections in the *BASIC-PLUS Language Manual.*

| Type | Function | Explanation |
|---|---|---|
| Mathematical | Y=ABS(X) | returns the absolute value of X. |
| | Y+ATN(X) | returns the arctangent of X, where X is in radians. |
| | Y=COS(X) | returns the cosine of X, where X is in radians. |
| | Y=EXP(X) | returns the value of e↑X, where e=2.71828. |
| | Y=FIX(X) | returns the truncated value of X, SGN(X)*INT(ABS(X)) |
| | Y=INT(X) | returns the greatest integer in X which is less than or equal to X. |
| | Y=LOG(X) | returns the natural logarithm of X, $\log_e X$. |
| | Y=LOG10(X) | returns the common logarithm of X, $\log_{10} X$. |
| | Y=PI | has a constant value of 3.14159. |
| | Y=RND | returns a random number between 0 and 1. |
| | Y=RND(X) | returns a random number between 0 and 1. |
| | Y=SGN(X) | returns the sign function of X, a value of 1 preceded by the sign of X. |
| | Y=SIN(X) | returns the sine of X, where X is in radians. |
| | Y=SQR(X) | returns the square root of X. |
| | Y=TAN(X) | returns the tangent of X, where X is in radians. |
| Print | Y%=POS(X%) | returns the current position of the print head for I/O channel X, 0 is the user's Teletype. (This value is imaginary for disk files.) |
| | Y$=TAB(X%) | moves print head to position X in the current print record, or is disregarded if the current position is beyond X. The first position is counted as 0. |

| Type | Function | Explanation |
|---|---|---|
| String | Y%=ASCII(A$) | returns the ASCII value of the first character in the string A$. |
| | Y$=CHR$(X%) | returns a character string having the ASCII value of X. Only one character is generated. |
| | Y$=CVT%$(I%) | maps integer into 2-character string, see Section 12.5. |
| | Y$=CVTF$(X) | maps floating-point number into 4- or 8-character string, see Section 12.5. |
| | Y%=CVT$%(A$) | maps first 2 characters of string A$ into an integer, see Section 12.5. |
| | Y=CVT$F(A$) | maps first 4 or 8 characters of string A$ into a floating-point number. See Section 12.5. |
| | Y$=CVT$$(A$,I%)[1] | converts string A$ to string Y$ according to value of I%. See Section 12.5. |
| | Y$=STRING$(N1%,N2%)[1] | creates string Y$ of length N1 and characters whose ASCII decimal value is N2. See Section 5.5. |
| | Y$=LEFT(A$,N%) | returns a substring of the string A$ from the first character to the Nth character (the leftmost N characters). |
| | Y$=RIGHT(A$,N%) | returns a substring of the string A$ from the Nth to the last character; the rightmost characters of the string starting with the Nth character. |
| | Y$=MID(A$,N1%,N2%) | returns a substring of the string A$ starting with the N1 and being N2 characters long (the characters between and including the N1 to N1+N2-1 characters). |
| | Y%=LEN(A$) | returns the number of characters in the string A$, including trailing blanks. |
| | Y%=INSTR(N1%,A$,B$) | indicates a search for the substring B$ within the string A$ beginning at character position N1. Returns a value 0 if B$ is not in A$, and the character position of B$ if B$ is found to be in A$ (character position is measured from the start of the string). |
| | Y$=SPACE$(N%) | indicates a string of N spaces, used to insert spaces within a character string. |
| | Y$=NUM$(N%) | indicates a string of numeric characters representing the value of N as it would be output by a PRINT statement. For example: NUM$(1.0000) = (space)1(space) and NUM$(-1.0000) = -1(space). |
| | Y=VAL(A$) | computes the numeric value of the string of numeric characters A$. If A$ contains any character not acceptable as numeric input with the INPUT statement, an error results. For example:<br><br>VAL("15")=15 |
| | Y$=XLATE(A$,B$) | translate A$ to the new string Y$ by means of the table string B$, see Section 12.7. |
| System | Y$=DATE$(0%) | returns the current date in the following format:<br><br>02-Mar-71 |

[1] These functions are not available prior to Version 5B(RSTS/E) systems.

| Type | Function | Explanation |
|---|---|---|
| System (Cont.) | Y$=DATE$(N%) | returns a character string corresponding to a calendar date as follows:<br>N=(day of year)+[(number of years since 1970)*1000]<br><br>DATE$(1) = "01-Jan-70"<br>DATE$(125) = "05-May-70"<br>DATE$(4125) = "05-May-74" |
| | Y$=TIME$(0%) | returns the current time of day as a character string as follows:<br><br>TIME$(0) = "05:30 PM" or "17:30 " |
| | Y$=TIME$(N%) | returns a string corresponding to the time at N minutes before midnight, for example:<br><br>TIME$(1) = "11:59 PM" or "23:59 "<br>TIME$(1440) = "12:00 AM" or "00:00 "<br>TIME$(721) = "11:59 AM" or "11:59 " |
| | Y=TIME(0%) | returns the clock time in seconds since midnight, as a floating point number. |
| | Y=TIME(1%) | returns the central processor time used by the current job in tenths of seconds. |
| | Y=TIME(2%) | returns the connect time (during which the user is logged into the system) for the current job in minutes. |
| | Y=TIME(3%)[1] | returns to Y the decimal number of kilo-core ticks (kct's) used by this job. See Section 8.8. |
| | Y=TIME(4%)[1] | returns to Y the decimal number of minutes of device time used by this job. See Section 8.8. |
| | Y%=STATUS[1] | returns to Y% the status of a channel as of the most recent OPEN statement executed in the program. See Section 12.3.5. |
| | Y%=BUFSIZ(N)[1] | returns to Y% the buffer size of the device or file open on Channel N. See Section 12.3.4. |
| | Y%=LINE | returns to Y% the line number of the statement being executed at the time of an interrupt. See Section 4.5. |
| | Y%=ERR | returns value associated with the last encountered error if an ON ERROR GOTO statement appears in the program. See Section 8.4. |
| | Y%=ERL | returns the line number at which the last error occurred if an ON ERROR GOTO statement appears in the program. See Section 8.4.3. |
| | Y%=SWAP%(N%) | causes a byte swap operation on the two bytes in the integer variable N%. |
| | Y$=RAD$(N%) | converts an integer value to a 3-character string and is used to convert from Radix-50 format back to ASCII. See Appendix D. |
| Matrix | MAT Y=TRN(X) | returns the transpose of the matrix X, see Section 7.6.2. |
| | MAT Y=INV(X) | returns the inverse of the matrix X, see Section 7.6.2. |
| | Y=DET | following an INV(X) function evaluation, the variable DET is equivalent to the determinant of X. |

[1] These functions are not available prior to Version 5B(RSTS/E) systems.

| Type | Function | Explanation |
|---|---|---|
| Matrix (Cont.) | Y%=NUM | following input of a matrix, NUM contains the number of rows input, or, in the case of a dimensional matrix, the number of elements entered. |
| | Y%=NUM2 | following input of a matrix, NUM2 contains the number of elements entered in that row. |
| Input/Output | Y%=RECOUNT | returns the number of characters read following every input operation. Used primarily with non-file structured devices. See Section 12.3.1. |

## A.4 SUMMARY OF BASIC-PLUS STATEMENTS

The following summary of statements available in the BASIC-PLUS language defines the general format for the statement as a line in a BASIC program. If more detailed information is needed, the reader is referred to the section(s) in the *BASIC-PLUS Language Manual* dealing with that particular statement.

**NOTE**

All section numbers refer to the *BASIC-PLUS Language Manual.*

In these definitions, elements in angle brackets are necessary elements of the statement. Elements in square brackets are necessary elements of which the statement may contain one. Elements in braces are optional elements of the statement.

Where the term line number ({*line number*}) is shown in braces, this statement can be used in immediate mode.

The various elements and their abbreviations are described below:

| | | |
|---|---|---|
| *variable* | or *var* | Any legal BASIC variable as described in A.1 or Section 2.5.2. |
| *line number* | | Any legal BASIC line number described in Section 2.2. |
| *expression* | or *exp* | Any legal BASIC expression as described in Section 2.5. |
| *message* | | Any combination of characters. |
| *condition* | or *cond* | Any logical condition as described in Section 3.5. |
| *constant* | | Any acceptable integer constant (need not contain a % character). |
| *argument(s)* | or *arg* | Dummy variable names. |
| *statement* | | Any legal BASIC-PLUS statement. |
| *string* | | Any legal string constant or variable as described in Section 5.1. |
| *protection* | | Any legal protection code as described in Section 9.1. |
| *value(s)* | | Any floating point, integer, or character string constant. |
| *list* | | The legal list for that particular statement. |
| *dimension(s)* | | One or two dimensions of a matrix, the maximum dimension(s) for that particular statement. |

```
                                                                       BASIC-PLUS Language
Statement Formats and Examples                                           Manual Section

  REM
    {line number} REM <message>                                             3.1
    {line number} {<statement>} ! <message>
              10       REM    THIS IS A COMMENT
              15       PRINT      !PERFORM A CR/LF

  LET
    {line number} {LET}<var>{,<var>,<var>. . .} = <exp>                     3.2
              55       LET A=40: B=22
              60       B,C,A=4.2          !MULTIPLE ASSIGNMENT

  DIM
    line number        DIM<var(dimension(s))>                               3.6.2
              10       DIM A(20), B$(5,10), C%(45)                            7.1

    line number        DIM #<constant>,<var(dimension(s)>=<constant>         11.1
              75       DIM #4, A$(100)=32,B(50,50)                           11.1

  RANDOMIZE                                                                 3.7.2
    line number        RANDOM{IZE}
              55       RANDOMIZE
              70       RANDOM

  IF-THEN, IF-GOTO
                                ┌ THEN <statement>    ┐
    line number        IF <cond>│ THEN<line number>   │                     3.5
                                └ GOTO<line number>   ┘
              55       IF A>B OR B>C THEN PRINT "NO"
              60       IF FNA(R) = B THEN 250
              95       IF L<X↑2 AND L<>0 GOTO 345

  IF-THEN-ELSE                                                              8.5
                                ┌ THEN <statement>    ┐
    line number        IF <cond>│ THEN<line number>   │ {ELSE<statement>   }
                                └ GOTO<line number>   ┘ {ELSE<line number> }
              30       IF B=A THEN PRINT "EQUAL" ELSE PRINT "NOT EQUAL"
              50       IF A>N THEN 200 ELSE PRINT A
              75       IF B == R THEN STOP ELSE 80

  FOR
    line number        FOR <var> = <exp> TO <exp> {STEP<exp>}               3.6.1
              20       FOR I=2 TO 40 STEP 2
              55       FOR N=A TO A+R

  FOR-WHILE, FOR-UNTIL                                                      8.6
                                                   ┌ WHILE ┐
    line number        FOR <var> = <exp> {STEP<exp>} │       │ <cond>
                                                   └ UNTIL ┘
              84       FOR I = 1 STEP 3 WHILE I<X
              74       FOR N = 2 STEP 4 UNTIL N>A OR N=B
              05       FOR B= 1 UNTIL B>10
```

| **Statement Formats and Examples** | **BASIC-PLUS Language Manual Section** |
|---|---|

```
NEXT                                                                   3.6.1
   line number   NEXT<var>
          25     NEXT I
          60     NEXT N

DEF, single line                                                       3.7.3
   line number   DEF FN<var>(arg) = <exp(arg)>                         5.5.1
          20     DEF FNA(X,Y,Z)=SQR(X↑2+Y↑2+Z↑2)                         6.4

DEF, multiple line                                                       8.1
   line number   DEF FN<var>(arg)
         <statements>
   line number   FN<var> =<exp>
   line number   FNEND
          10     DEF FNF(M)      !FACTORIAL FUNCTION
          20     IF M=1 THEN FNF=1 ELSE FNF=M*FNF(M-1)
          30     FNEND

GOTO                                                                     3.4
   line number   GOTO <line number>
         100     GOTO 50

ON-GOTO
   line number   ON <exp> GOTO <list of line numbers>                    8.2
          75     ON X GOTO 95, 150, 45, 200

GOSUB
   line number   GOSUB <line number>                                   3.8.1
          90     GOSUB 200

ON-GOSUB
   line number   ON <exp> GOSUB <list of line numbers>                   8.3
          85     ON FNA(M) GOSUB 200, 250, 400, 375

RETURN
   line number   RETURN                                                3.8.2
         375     RETURN

CHANGE                                                                   5.2
  {line number}  CHANGE [<array name>]   TO  [<string var> ]
                        [<string var>]       [<array name>]
          25     CHANGE A$ TO X
          70     CHANGE M TO R$
          75     CHANGE B TO B$

OPEN                                                                     9.2
  {line number}  OPEN<string> {FOR [INPUT ]} AS FILE <exp>             9.2.1
                                   [OUTPUT]
         {,RECORDSIZE<exp>} {,CLUSTERSIZE<exp>} {,MODE<exp>}           9.2.2
          10     OPEN "PP:" FOR OUTPUT AS FILE B1
          20     OPEN "FOO" AS FILE 3
          30     OPEN "DT4:DATA.TR" FOR INPUT AS FILE 10
```

| **Statement Formats and Examples** | | **BASIC-PLUS Language Manual Section** |
|---|---|---|
| CLOSE | | 9.3 |
| {*line number*} | CLOSE *<list of exp>* | |
| 100 | CLOSE 2 | |
| 255 | CLOSE 10, 4, N1 | |
| | | 3.3.1 |
| READ | | 5.3 |
| *line number* | READ *<list of variables>* | 6.3 |
| 25 | READ A, B$, C%, F1, R2, B(25) | 10.1 |
| DATA | | 3.3.1 |
| *line number* | DATA *<list of values>* | 5.3 |
| 300 | DATA 4.3, “STRING”,85,49,75.04,10 | 6.3 |
| RESTORE | | 3.3.1 |
| *line number* | RESTORE | 10.2 |
| 125 | RESTORE | |
| PRINT | | 3.3.2 |
| | | 5.4 |
| {*line number*} | PRINT {{#*<exp>*,}*<list>*} | 6.3 |
| 25 | PRINT !GENERATES CR/LF | 10.3 |
| 75 | PRINT “BEGINNING OF OUTPUT”;I,A*I | 10.3.1 |
| 45 | PRINT #4, “OUTPUT TO DEVICE”FNM(A)↑2;B;A | 10.3.2 |
| PRINT USING | | |
| {*line number*} | PRINT {#*<exp>*, }USING *<string>*,*<list>* | 10.3.3 |
| 54 | PRINT USING “##.##”,A | |
| 55 | PRINT #3, USING“\\###.## \\##↑↑↑↑”,“A=”,A,“B=”,B | |
| 56 | PRINT #7, USING B$,A,B,C | |
| INPUT | | 3.3.3 |
| {*line number*} | INPUT {#*<exp>*,}*<list>* | 5.3 |
| 25 | INPUT “TYPE YOUR NAME ”,A$ | 10.4 |
| 55 | INPUT #8, A, N, B$ | 10.4.1 |
| INPUT LINE | | 5.3 |
| {*line number*} | INPUT LINE {#*<exp>*,}*<string>* | |
| 40 | INPUT LINE R$ | |
| 75 | INPUT LINE #1, E$ | |
| NAME-AS | | 9.4 |
| {*line number*} | NAME *<string>* AS *<string>* | |
| 455 | NAME “NONAME” AS “FILE1<48>” | |
| 270 | NAME “DT4:MATRIX” AS “MATA1<48>” | |
| KILL | | 9.5 |
| {*line number*} | KILL*<string>* | |
| 45 | KILL “NONAME” | |

| **Statement Formats and Examples** | | **BASIC-PLUS Language Manual Section** |
|---|---|---|
| ON ERROR GOTO | | 8.4 |
| *line number* | ON ERROR GOTO {<*line number*>} | |
| 10 | ON ERROR GOTO 500 | |
| 525 | ON ERROR GOTO !DISABLES ERROR ROUTINE | |
| 526 | ON ERROR GOTO 0 !DISABLES ERROR ROUTINE | |
| RESUME | | 8.4.1 |
| *line number* | RESUME {<*line number*>} | |
| 1000 | RESUME !OR RESUME 0 ARE EQUIVALENT | |
| 655 | RESUME 200 | |
| CHAIN | | |
| *line number* | CHAIN <*string*> {<*exp*>} | 9.6 |
| 375 | CHAIN "PROG2" | |
| 500 | CHAIN "PROG3" 75 | |
| 600 | CHAIN "PROG3" A | |
| STOP | | 3.9 |
| *line number* | STOP | |
| 75 | STOP | |
| END | | 3.9 |
| *line number* | END | |
| 545 | END | |

**Matrix Statements**

| | | |
|---|---|---|
| MAT READ | | 7.2 |
| *line number* | MAT READ <*list of matrices*> | |
| 55 | DIM A(20), B$(32), C%(15,10) | |
| 90 | MAT READ A, B$(25), C% | |
| MAT PRINT | | 7.3 |
| {*line number*} | MAT PRINT{#<*exp*>,}<*matrix name*> | |
| 10 | DIM A(20), B(15,20) | |
| 90 | MAT PRINT A; !PRINT 10*10 MATRIX, PACKED | |
| 95 | MAT PRINT B(10,5), !PRINT 10*5 MATRIX, FIVE | |
| | !ELEMENTS PER LINE | |
| 97 | MAT PRINT #2, A; !PRINT ON OUTPUT CHANNEL 2 | |
| MAT INPUT | | 7.4 |
| {*line number*} | MAT INPUT {#<*exp*>,}<*list of matrices*> | |
| 10 | DIM B$(40), F1%(35) | |
| 20 | OPEN "DT3:FOO" FOR INPUT AS FILE 3 | |
| 30 | MAT INPUT #3, B4, F1% | |

| | **Statement Formats and Examples** | | **BASIC-PLUS Language Manual Section** |
|---|---|---|---|
| **Matrix Statements (Cont.)** | | | |
| MAT Initialization | | | 7.5 |
| {*line number*} | MAT <*matrix name*> = $\left[\begin{array}{c}\text{ZER}\\\text{CON}\\\text{IDN}\end{array}\right]$ {*dimension(s)*} | | |
| 10 | DIM B(15,10), A(10), C%(5) | | |
| 15 | MAT C% = CON | !ALL ELEMENTS OF C%(I)=1 | |
| 20 | MAT B = IDN(10,10) | !IDENTITY MATRIX 10*10 | |
| 95 | MAT B = ZER(N,M) | !CLEARS AN N BY M MATRIX | |
| **Statement Modifiers** (can be used in immediate mode) | | | |
| IF | | | 8.7.1 |
| <*statement*> | IF <*condition*> | | |
| 10 | PRINT X IF X<>0 | | |
| UNLESS | | | 8.7.2 |
| <*statement*> | UNLESS <*condition*> | | |
| 45 | PRINT A UNLESS A=0 | | |
| FOR | | | 8.7.3 |
| <*statement*> | FOR <*var*> = <*exp*> TO <*exp*> {STEP<*exp*> | | |
| 75 | LET B$(I) = "PDP-11" FOR I = 1 TO 25 | | |
| 80 | READ A(I) FOR I=2 TO 8 STEP 2 | | |
| WHILE | | | 8.7.4 |
| <*statement*> WHILE <*condition*> | | | |
| 10 | LET A(I) = FNX(I) WHILE I<45.5 | | |
| UNTIL | | | 8.7.5 |
| <*statement*> | UNTIL <*condition*> | | |
| 115 | IF B 0 THEN A(I)=B UNTIL I > 5 | | |
| **System Statements** | | | |
| <*line number*> | SLEEP <*expression*> | | 8.8 |
| 100 | SLEEP(20) !DISMISS JOB FOR 20 SEC. | | |
| <*line number*> | WAIT <*expression*> | | 8.8 |
| 525 | WAIT(A%+5) !WAIT A%+5 SEC. FOR INPUT | | |
| **Record I/O Statements** | | | |
| <*line number*> | LSET<*string var*>{,<*string var*>}= <*string*> | | 12.4.3 |
| 90 | LSET B$="XYZ" | | |
| <*line number*> | RSET<*string var*> {,<*string var*>=<*string*> | | 12.4.3 |
| 250 | RSET C$="67890" | | |
| <*line number*> | FIELD #<*expr*>,<*expr*> AS <*string var*> {,<*expr*> AS <*string var*>} | | 12.4.2 |
| 75 | FIELD #2%, 10% AS A$, 20% AS B$ | | |

| **Statement Formats and Examples** | | **BASIC-PLUS Language Manual Section** |
|---|---|---|
| **Record I/O Statements (Cont.)** | | |
| *<line number>* | GET #*<expr>*{,RECORD*<expr>*} | 12.3 |
| 100 | GET #1%,RECORD 99% | |
| *<line number>* | PUT #*<expr>*{,RECORD*<expr>*}{,COUNT*<expr>*} | 12.3 |
| 500 | PUT # 1%, COUNT 80% | |
| *<line number>* | UNLOCK #*<expr>* | 10.5.1 |
| 700 | UNLOCK #3% | |

# APPENDIX B
# BASIC-PLUS COMMAND SUMMARY

| Command | Explanation | Section in RSTS/E System User's Guide |
|---|---|---|
| APPEND | Used to include contents of a previously saved source program in current program. | 9.1.5 |
| ASSIGN | Used to reserve an I/O device for the use of the individual issuing the command. The specified device can then be given commands only from the terminal which issued the ASSIGN. Also establishes a logical name for a device, establishes an account for the @ character, and assigns a default protection code. | 5.1 |
| ATTACH | Attaches a detached job to the current terminal. | Chapter 14 |
| BYE | Indicates to RSTS that a user wishes to leave the terminal. Closes and saves any files remaining open for that user. | Chapter 14 |
| CAT<br>CATALOG | Returns the user's file directory. Unless another device is specified following the term CAT or CATALOG, the disk is the assumed device. | 8.9 |
| CCONT | For privileged users. Same as CONT command but detaches job from terminal. | 10.2.3 |
| COMPILE | Allows the user to store a compiled version of his BASIC program. The file is stored on disk with the current name and the extension .BAC. Or, a new file name can be indicated and the extension .BAC will still be appended. | 8.4.3 |
| CONT | Allows the user to continue execution of the program currently in core following the execution of a STOP statement. | 10.2.2 |
| DEASSIGN | Used to release the specified device for use by others. If no particular device is specified, all devices assigned to that terminal are released. An automatic DEASSIGN is performed when the BYTE command is given. Also releases any logical name for a device. | 5.2, 5.5, 5.7 |

| Command | Explanation | Section in RSTS/E System User's Guide |
|---|---|---|
| DELETE | Allows the user to remove one or more lines from the program currently in core. Following the word DELETE the user types the line number of the single line to be deleted or two line numbers separated by a dash (-) indicating the first and last line of the section of code to be removed. Several single lines or line sections can be indicated by separating the line numbers, or line number pairs, with a comma. | 9.1.2 |
| HELLO | Indicates to RSTS that a user wishes to log onto the system. Allows the user to input project-programmer number and password. Also attaches a detached job to the current terminal or changes accounts without having to log off the system. | Chapter 14 |
| KEY | Used to re-enable the echo feature on the user terminal following the issue of a TAPE command. Enter with LINE FEED or ESCAPE key. | 5.8.2 |
| LENGTH | Returns the length of the user's current program in core, in 1K increments. | 8.7.2 |
| LIST | Allows the user to obtain printed listing at the user terminal of the program currently in core, or one or more lines of that program. The word LIST by itself will cause the listing of the entire user program. LIST followed by one line number will list that line; and LIST followed by two line numbers separated by a dash (-) will list the lines between and including the lines indicated. Several single lines or line sections can be indicated by separating the line numbers, or line number pairs, with a comma. | 9.1.1 |
| LISTNH | Same as LIST, but does not print header containing the program name and current date. | 9.1.1 |
| LOGIN | Same as HELLO. | Chapter 14 |
| NEW | Clears the user's area in core and allows the user to input a new program from the terminal. A program name can be indicated following the word NEW or when the system requests it. | 8.1.1 |
| OLD | Clears the user's area in core and allows the user to recall a saved program from a storage device. The user can indicate a program name following the word OLD or when the system requests it. If no device name is given, the file is assumed to | 8.3 |

| **Command** | **Explanation** | **Section in RSTS/E System User's Guide** |
|---|---|---|
| OLD (Cont.) | be on the system disk. A device specification without a filename will cause a program to be read from an input-only device (such as high-speed reader, card reader). | 8.3 |
| REASSIGN | Transfers control of a device to another job. | 5.3 |
| RENAME | Causes the name of the program currently in core to be changed to the name specified after the word RENAME. | 8.5 |
| REPLACE | Same as SAVE, but allows the user to substitute a new program with the same name for an old program, erasing the old program. | 8.6 |
| RUN | Allows the user to begin execution of the program currently in core. The word RUN can be followed by a file name in which case the file is loaded from the system disk, compiled, and run alternatively, the device and file name can be indicated if the file is not on the system disk. A device specification without a file name will cause a program to be read from an input only device (such as high-speed reader, card reader). | 8.4.1 |
| RUNNH | Causes execution of the program currently in memory but header information containing the program name and current date is not printed. If a filename is used, the command is executed as if no filename were given. | 8.4.1 |
| SAVE | Causes the program currently in core to be saved on the system disk under its current file name with the extension .BAS. Where the word SAVE is followed by a file name or a device and a file name, the program in core is saved under the name given and on the device specified. A device specification without a file name will cause the program to be output to any output only device (line printer, high-speed punch). | 8.2 |
| SCALE | Sets the scale factor to a designated value or prints the value(s) currently in effect if no value is designated. | 8.10 |
| TAPE | Used to disable the echo feature on the user terminal while reading paper tape via the low-speed reader. | 5.8.1 |

| Command | Explanation | Section in RSTS/E System User's Guide |
|---|---|---|
| UNSAVE | The word UNSAVE is followed by the file name and, optionally, the extension of the file to be removed. The UNSAVE command cannot remove files without an extension. If no extension is specified, the source (.BAS) file is deleted. If no device is specified, the disk is assumed. | 9.1.4 |

**Special Control Character Summary**

| | | |
|---|---|---|
| CTRL/C | Causes the system to return to BASIC command mode to allow for issuing of further commands or editing. Echoes on terminal as ↑C. | 5.9.1 |
| CTRL/O | Used as a switch to suppress/enable output of a program on the user terminal. Echoes as ↑O. | 5.9.2 |
| CTRL/Q | When generated by a device on which a CTRL/S has interrupted output, causes computer to resume output at the next character. | 5.9.3 |
| CTRL/S | When generated by a device for which SCOPE characteristics are set, interrupts computer output on the device until either CTRL/Q or another character is generated. | 5.9.3 |
| CTRL/U | Deletes the current typed line, echoes as ↑U and performs a carriage return/line feed. | 9.1.3 |
| CTRL/Z | Used as an end-of-file character. | 5.9.4 |
| ESCape or ALT MODE Key | Enters a typed line to the system, echoes on the user terminal as a $ character and does not cause a carriage return/line feed. | 5.9.6 |
| LINE FEED Key | Used to continue the current logical line on an additional physical line. Performs a carriage return/line feed operation. | 9.2.2.2 |
| RETURN Key | Enters a typed line to the system, results in a carriage return/line feed operation at the user terminal. | 5.9.5 |
| RUBOUT Key | Deletes the last character typed on that physical line. Erased characters are shown on the teleprinter between back slashes. | 9.1.3 |
| TAB or CTRL/I | Performs a tabulation to the next of nine tab stops (eight spaces apart) which form the terminal printing line. | 9.2.2.3 |
| CTRL/L | Generates FORM FEED character and results in four line feed operations at the user terminal. | 9.2.2 |

# APPENDIX C
# ERROR MESSAGES

Messages in RSTS/E are generated for BASIC-PLUS errors[1] and RSTS/E errors. To avoid confusion, both types of messages are called RSTS/E error messages and are described as one set. The BASIC-PLUS errors cover compiler and run time conditions such as a violation of the syntax rules (SYNTAX ERROR) and referencing an element of an array beyond the defined limits (SUBSCRIPT OUT OF RANGE). The RSTS/E errors involve operating system conditions such as failing to locate the file or account specified (CAN'T FIND FILE OR ACCOUNT) and requesting the hardware to perform a function for which it is not ready (DEVICE HUNG OR WRITE LOCKED).

In most cases, if no error trapping is being done (that is, an ON ERROR GOTO statement is not in effect), BASIC-PLUS stops running the program. It prints the error message and the line number of the BASIC-PLUS statement that was being executed when the error occurred. The following sample printout shows the procedure.

```
10      OPEN 'Z' FOR INPUT AS FILE 1%
RUNNH
?CAN'T FIND FILE OR ACCOUNT AT LINE 10

READY
```

As the READY message indicates, control returns to the system.

An exception to this procedure occurs when an INPUT statement is being executed at the job's console terminal and error trapping is not in effect. The system generates the error message and executes the statement again as shown in the sample printout below.

```
10      ON ERROR GOTO 0 \ INPUT 'INTEGER VALUE';A%
RUNNH
INTEGER VALUE? C
%DATA FORMAT ERROR AT LINE 10
INTEGER VALUE?
```

With error trapping disabled at line 10, an invalid response to the INPUT statement causes the system to print the error message, clear the error condition, and execute the statement again.

Associated with each message is an error variable called ERR. Whenever an error occurs with trapping in effect, the system checks the error variable which is a decimal number in the range 0 to 127. An error with a number between 1 and 70 causes the system to transfer control to the line number indicated in the ON ERROR GOTO statement. The system does not print the error message. The user program is able to check the ERR variable and perform a recovery procedure. If the error number is between 71 and 127, the system does not transfer control to the recovery routine but prints the message and returns control to the system. (Error number 0 is reserved to identify the system installation name.)

Because a BASIC-PLUS program can recover from certain errors, this appendix lists errors in two categories – recoverable and non-recoverable. The recoverable error messages are listed in ascending order of their related error numbers. A program can use these error numbers to differentiate errors. Non-recoverable errors are in alphabetical order without error numbers because a program can not use these numbers in an error handling routine.

[1]Different messages are generated while a job is operating under run-time systems other than BASIC-PLUS. Such run-time systems are those for COBOL and FORTRAN-IV. For these error messages, consult the appropriate User's Guides.

The first character position of each.message indicates the severity of the error. Table C-1 describes this standard.

**Table C-1 Severity Standard in Error Messages**

| Character | Severity | Meaning |
|---|---|---|
| % | Warning | Execution of the program can continue but may not generate the expected results. |
| ? | Fatal | Execution cannot continue unless the user removes the cause of the error. |
| | Information | A message beginning with neither a question mark nor a percent is for information only. |

The severity indication is useful for utility programs such as BATCH which examines system output.

In the descriptions of error messages, certain abbreviations, as shown in Table C-2, denote special characteristics of the error.

**Table C-2 Special Abbreviations for Error Descriptions**

| Abbreviation | Meaning |
|---|---|
| (C) | Continue. If an ON ERROR GOTO statement is not in effect, execution continues but with the conditions described. |
| (SPR) | Software Performance Report. This error should occur only under the conditions described. If it occurs under any other conditions, the user should file an SPR with DIGITAL and document the conditions under which the error occurred. |

An error whose description is accompanied by the abbreviation (C) indicates an exception to the error trapping procedure. If such an error occurs in a program with no error trapping in effect, BASIC-PLUS prints the error message and line number but continues running the program. The following sample printout shows the procedure.

```
100     ON ERROR GOTO 0 \ A% = 32768.
200     PRINT A%
RUNNH
%INTEGER ERROR AT LINE 100
 0

READY
```

The INTEGER ERROR is generated at line 100 by the attempt to compute a value outside the range for integers. After the error message is printed, processing continues but with the conditions described in the error meaning. 0 is substituted for the erroneously computed value.

The number of RSTS/E error messages is restricted to 127. Because of this restriction, certain error messages have multiple meanings. The specific meaning of an error message depends on the operation being performed when the error condition occurs. For example, if the system attempts a file access and the designated file can not be located, RSTS/E generates the CAN'T FIND FILE OR ACCOUNT error (ERR=5). That same error condition, however, applies to other, generically similar access operations. Thus, if a program attempts to send a message to another program and the proper entry is not found in the system table of eligible receivers, RSTS/E returns error number 5. Though the second failure does not involve a file access error, it too is classified as an access failure.

Certain RSTS/E errors, although classified as user recoverable, are not capable of being trapped by a program. Table C-3 lists such errors.

**Table C-3 Non-Trappable Errors in Recoverable Class**

| ERR | Message Printed |
|---|---|
| 34 | RESERVED INSTRUCTION TRAP |
| 36 | SP (R6) STACK OVERFLOW |
| 37 | DISK ERROR DURING SWAP |
| 38 | MEMORY PARITY FAILURE |

These errors involve special conditions which a user program cannot control and which ought not to occur on a normal system. For example, the DISK ERROR DURING SWAP error indicates a hardware problem. The system does not return control to the program. The error condition itself, however, can be either transient or recurring. Such errors should be brought to the attention of the system manager for further investigation. These errors are recoverable in the strict sense that the monitor can take corrective action but the BASIC-PLUS run-time system does not return control to the user program.

## C.1 USER RECOVERABLE

| ERR | Message Printed | Meaning |
|---|---|---|
| 0 | (system installation name) | The error code 0 is associated with the system installation name and is used by system programs to print identification lines. |
| 1 | ?Bad directory for device | The directory of the device referenced is in an unreadable format. The magtape label format on tape differs from the system-wide default format, the current job default format, or the format specified in the OPEN statement. Use the ASSIGN command to set the correct format default or change the format specification in the MODE option of the OPEN statement. |
| 2 | ?Illegal file name | The filename specified is not acceptable. It contains unacceptable characters or the filename specification format has been violated. The CCL command to be added begins with a number or contains a character other than A through Z, 0 through 9 and commercial at (@). |
| 3 | ?Account or device in use | Reassigning or dismounting of the device cannot be done because the device is open or has one or more open files. The account to be deleted has one or more files and must be zeroed before being deleted. The run time system to be deleted is currently loaded in memory and in use. Output to a pseudo keyboard cannot be done unless the device is in KB wait state. An echo control field cannot be declared while another field is currently active. The CCL command to be added already exists. |

| ERR | Message Printed | Meaning |
|---|---|---|
| 4 | ?No room for user on device | Storage space allowed for the current user on the device specified has been used or the device as a whole is too full to accept further data. |
| 5 | ?Can't find file or account | The file or account number specified was not found on the device specified. The CCL command to be deleted does not exist. |
| 6 | ?Not a valid device | The device specification supplied is not valid for one of the following reasons. The unit number or its type is not configured on the system. The specification is logical and untranslatable because a physical device is not associated with it. |
| 7 | ?I/O channel already open | An attempt was made to open one of the twelve I/O channels which had already been opened by the program. (SPR) |
| 8 | ?Device not available | The specified device exists on the system but a user's attempt to ASSIGN or OPEN it is prohibited for one of the following reasons. The device is currently reserved by another job. The device requires privileges for ownership and the user does not have privilege. The device or its controller has been disabled by the system manager. The device is a keyboard line for pseudo keyboard use only. |
| 9 | ?I/O channel not open | Attempt to perform I/O on one of the twelve channels which has not been previously opened in the program. |
| 10 | ?Protection violation | The user was prohibited from performing the requested operation because the kind of operation was illegal (such as input from a line printer) or because the user did not have the privileges necessary (such as deleting a protected file). |
| 11 | ?End of file on device | Attempt to perform input beyond the end of a data file; or a BASIC source file is called into memory and is found to contain no END statement. |
| 12 | ?Fatal system I/O failure | An I/O error has occurred on the system level. The user has no guarantee that the last operation has been performed. This error is caused by hardware condition. Report such occurrences to the system manager. (See the discussion at beginning of appendix.) |
| 13 | ?User data error on device | One or more characters may have been transmitted incorrectly due to a parity error, bad punch combination on a card, or similar error. |
| 14 | ?Device hung or write locked | User should check hardware condition of device requested. Possible causes of this error include a line printer out of paper or high-speed reader being off-line. |
| 15 | ?Keyboard WAIT exhausted | Time requested by WAIT statement has been exhausted with no input received from the specified keyboard. |
| 16 | ?Name or account now exists | An attempt was made to rename a file with the name of a file which already exists, or an attempt was made by the system manager to insert an account number which is already within the system. |

| ERR | Message Printed | Meaning |
|---|---|---|
| 17 | ?Too many open files on unit | Only one open DECtape output file is permitted per DECtape drive. Only one open file per magtape drive is permitted. |
| 18 | ?Illegal SYS( ) usage | Illegal use of the SYS system function. |
| 19 | ?Disk block is interlocked | The requested disk block segment is already in use (locked) by some other user. |
| 20 | ?Pack IDs don't match | The identification code for the specified disk pack does not match the identification code already on the pack. |
| 21 | ?Disk pack is not mounted | No disk pack is mounted on the specified disk drive. |
| 22 | ?Disk pack is locked out | The disk pack specified is mounted but temporarily disabled. |
| 23 | ?Illegal cluster size | The specified cluster size is unacceptable. The cluster size must be a power of 2. For a file cluster, the size must be equal to or greater than the pack cluster size and must not be greater than 256. For a pack cluster, the size must be equal to or greater than the device cluster size and must not be greater than 16. The device cluster size is fixed by type. |
| 24 | ?Disk pack is private | The current user does not have access to the specified private disk pack. |
| 25 | ?Disk pack needs 'CLEANing' | Non-fatal disk mounting error; use the CLEAN operation in UTILTY. |
| 26 | ?Fatal disk pack mount error | Fatal disk mounting error. Disk cannot be successfully mounted. |
| 27 | ?I/O to detached keyboard | I/O was attempted to a hung up dataset or to the previous, but now detached, console keyboard for the job. |
| 28 | ?Programmable ↑C trap | A CTRL/C combination was typed while an ON ERROR GOTO statement was in effect and programmable CTRL/C trapping was enabled. |
| 29 | ?Corrupted file structure | Fatal error in CLEAN operation. |
| 30 | ?Device not file structured | An attempt is made to access a device, other than a disk, DECtape, or magtape device, as a file-structured device. This error occurs, for example, when the user attempts to gain a directory listing of a non-directory device. |
| 31 | ?Illegal byte count for I/O | The buffer size specified in the RECORDSIZE option of the OPEN statement or in the COUNT option of the PUT statement is not a multiple of the block size of the device being used for I/O, or is illegal for the device. An attempt is made to run a compiled file which has improper size due to incorrect transfer procedure. |
| 32 | ?No buffer space available | The user accesses a file and the monitor requires one small buffer to complete the request but one is not currently available. If the program is sending messages, two conditions are possible. The first occurs when a program sends a message and the receiving program has exceeded the pending message limit. The second occurs when a sending program attempts to send a message and a small buffer is not available for the operation. |

| ERR | Message Printed | Meaning |
|---|---|---|
| 33 | ?UNIBUS timeout fatal trap | This hardware error occurs when an attempt is made to address non-existent memory or an odd address using the PEEK function. An occurrence of this error message in any other case is cause for an SPR. |
| 34 | ?Reserved instruction trap | An attempt is made to execute an illegal or reserved instruction or an FPP instruction when floating point hardware is not available. (See discussion at beginning of appendix.) |
| 35 | ?Memory management violation | This hardware error occurs when an illegal Monitor address is specified using the PEEK function. Generation of the error message in situations other than using PEEK is cause for an SPR. |
| 36 | ?SP (R6) stack overflow | An attempt to extend the hardware stack beyond its legal size is encountered. (See discussion at beginning of appendix.) (SPR) |
| 37 | ?Disk error during swap | A hardware error occurs when a user's job is swapped into or out of memory. The contents of the user's job area are lost but the job remains logged into the system and is reinitialized to run the NONAME program. Report such occurrences to the system manager. (See discussion at beginning of appendix.) |
| 38 | ?Memory parity failure | A parity error was detected in the memory occupied by this job. (See discussion at beginning of appendix.) |
| 39 | ?Magtape select error | When access to a magtape drive was attempted, the selected unit was found to be off line. |
| 40 | ?Magtape record length error | When performing input from magtape, the record on magtape was found to be longer than the buffer designated to handle the record. |
| 41 | ?Non-res run-time system | The run time system referenced has not been loaded into memory and is therefore non-resident. |
| 42 | ?Virtual buffer too large | Virtual core buffers must be 512 bytes long. |
| 43 | ?Virtual array not on disk | A non-disk device is open on the channel upon which the virtual array is referenced. |
| 44 | ?Matrix or array too big | In-core array size is too large. |
| 45 | ?Virtual array not yet open | An attempt was made to use a virtual array before opening the corresponding disk file. |
| 46 | ?Illegal I/O channel | Attempt was made to open a file on an I/O channel outside the range of the integer numbers 1 to 12. |
| 47 | ?Line too long | Attempt to input a line longer than 255 characters (which includes any line terminator). Buffer overflows. |
| 48 | %Floating point error | Attempt to use a computed floating point number outside the range 1E-38 $<n<$ 1E38 excluding zero. If no transfer to an error handling routine is made, zero is returned as the floating point value. (C) |

| ERR | Message Printed | Meaning |
|---|---|---|
| 49 | %Argument too large in EXP | Acceptable arguments are within the approximate range -89<arg<+88. The value returned is zero. (C) |
| 50 | %Data format error | A READ or INPUT statement detected data in an illegal format. For example, 1..2 is an improperly formed number, and 1.3 is an improperly formed integer, and X" is an illegal string. (C) |
| 51 | %Integer error | Attempt to use a computed integer outside the range -32768<n<32767. For example, an attempt is made to assign to an integer variable a floating point number outside the integer range. If no transfer to an error handling routine is made, zero is returned as the integer value. (C) |
| 52 | ?Illegal number | Integer overflow or underflow or floating point overflow. The range for integers is -32768 to +32767; for floating point numbers, the upper limit is 1E38. (For floating point underflow, the FLOATING POINT ERROR (ERR=48) is generated.) |
| 53 | %Illegal argument in LOG | Negative or zero argument to LOG function. Value returned is the argument as passed to the function. (C) |
| 54 | %Imaginary square roots | Attempt to take square root of a number less than zero, The value returned is the square root of the absolute value of the argument. (C) |
| 55 | ?Subscript out of range | Attempt to reference an array element beyond the number of elements created for the array when it was dimensioned. |
| 56 | ?Can't invert matrix | Attempt to invert a singular or nearly singular matrix. |
| 57 | ?Out of data | The DATA list was exhausted and a READ requested additional data. |
| 58 | ?ON statement out of range | The index value in an ON-GOTO or ON-GOSUB statement is less than one or greater than the number of line numbers in the list. |
| 59 | ?Not enough data in record | An INPUT statement did not find enough data in one line to satisfy all the specified variables. |
| 60 | ?Integer overflow, FOR loop | The integer index in a FOR loop attempted to go beyond 32766 or below -32767. |
| 61 | %Division by 0 | Attempt by the user program to divide some quantity by zero. If no transfer is made to an error handler routine, a 0 is returned as the result. (C) |
| 62 | ?No run-time system | The run-time system referenced has not been added to the system list of run time systems. |
| 63 | ?FIELD overflows buffer | Attempt to use FIELD to allocate more space than exists in the specified buffer. |
| 64 | ?Not a random access device | Attempt to perform random access I/O to a non-random access device. |

| ERR | Message Printed | Meaning |
|---|---|---|
| 65 | ?Illegal MAGTAPE ( ) usage | Improper use of the MAGTAPE function. |
| 66 | ?Missing special feature | User program employs a BASIC-PLUS feature not present on the given installation. |
| 67 | ?Illegal switch usage | A CCL command contains an error in an otherwise valid CCL switch. (For example, the /SI:n switch was used without a value for n or a colon; or more than one of the same type of CCL switch was specified.) A file specification switch is not the last element in a file specification or is missing a colon or an argument. |

## C.2 NON-RECOVERABLE

| Message Printed | Meaning |
|---|---|
| ?Arguments don't match | Arguments in a function call do not match, in number or in type, the arguments defined for the function. |
| ?Bad line number pair | Line numbers specified in a LIST or DELETE command were formatted incorrectly. |
| ?Bad number in PRINT-USING | Format specified in the PRINT-USING string cannot be used to print one or more values. |
| ?Can't compile statement | |
| ?Can't CONTinue | Program was stopped or ended at a spot from which execution cannot be resumed. |
| ?Data type error | Incorrect usage of floating-point, integer, or character string format variable or constant where some other data type was necessary. |
| ?DEF without FNEND | A second DEF statement was encountered in the processing of a user function without an FNEND statement terminating the first user function definition. |
| ?End of statement not seen | Statement contains too many elements to be processed correctly. |
| ?Execute only file | Attempt was made to add, delete or list a statement in a compiled (.BAC) format file. |
| ?Expression too complicated | This error usually occurs when parentheses have been nested too deeply. The depth allowable is dependent on the individual expression. |
| ?File exists-RENAME/REPLACE | A file of the name specified in a SAVE command already exists. In order to save the current program under the name specified, use REPLACE, or use RENAME followed by SAVE. |
| ?FNEND without DEF | An FNEND statement was encountered in the user program without a previous function call having been executed. |

| Message Printed | Meaning |
|---|---|
| ?FNEND without function call | A FNEND statement was encountered in the user program without a previous DEF statement being seen. |
| ?FOR without NEXT | A FOR statement was encountered in the user program without a corresponding NEXT statement to terminate the loop. |
| ?Illegal conditional clause | Incorrectly formatted conditional expression. |
| ?Illegal DEF nesting | The range of one function definition crosses the range of another function definition. |
| ?Illegal dummy variable | One of the variables in the dummy variable list of user-defined function is not a legal variable name. |
| ?Illegal expression | Double operators, missing operators, mismatched parentheses, or some similar error has been found in an expression. |
| ?Illegal FIELD variable | The FIELD variable specified is unacceptable. |
| ?Illegal FN redefinition | Attempt was made to redefine a user function. |
| ?Illegal function name | Attempt was made to define a function with a function name not subscribing to the established format. |
| ?Illegal IF statement | Incorrectly formatted IF statement. |
| ?Illegal in immediate mode | User issued a statement for execution in immediate mode which can only be performed as part of a program. |
| ?Illegal line number(s) | Line number reference outside the range 1<n<32767. |
| ?Illegal mode mixing | String and numeric operations cannot be mixed. |
| ?Illegal statement | Attempt was made to execute a statement that did not compile without errors. |
| ?Illegal symbol | An unrecognizable character was encountered. For example, a line consisting of a # character. |
| ?Illegal verb | The BASIC verb portion of the statement cannot be recognized. |
| %Inconsistent function usage | A function is defined with a certain number of arguments but is elsewhere referenced with a different number of arguments. Fix the reference to match the definition and reload the program to reset the function definition. |
| %Inconsistent subscript use | A subscripted variable is being used with a different number of dimensions from the number with which it was originally defined. |
| x(y)K of memory used | Message printed by the LENGTH command. The value for x is the current size, to the nearest 1K-word increment, of the program in memory. The value for y is the size to which the program can expand, given the run time system being used and the job's private memory size maximum set by the system manager. |

| Message Printed | Meaning |
|---|---|
| ?Literal string needed | A variable name was used where a numeric or character string was necessary. |
| ?Matrix dimension error | Attempt was made to dimension a matrix to more than two dimensions, or an error was made in the syntax of a DIM statement. |
| ?Matrix or array without DIM | A matrix or array element was referenced beyond the range of an implicitly dimensioned matrix. |
| ?Maximum memory exceeded | During an OLD operation, the job's private memory size maximum was reached. While running a program, the system required more memory for string or I/O buffer space and the job's private memory size maximum or the system maximum (16K words for BASIC-PLUS) was reached. |
| ?Modifier error | Attempt to use one of the statement modifiers (FOR, WHILE, UNTIL, IF, or UNLESS) incorrectly. An OPEN statement modifier, such as a RECORDSIZE, CLUSTERSIZE, FILESIZE, or MODE option, is not in the correct order. |
| ?NEXT without FOR | A NEXT statement was encountered in the user program without a previous FOR statement having been seen. |
| ?No logins | Message printed if the system is full and cannot accept additional users or if further logins are disabled by the system manager. |
| ?Not enough available memory | An attempt is made to load a non-privileged compiled program which is too large to run, given the job's private memory size maximum. The program must be made privileged to allow it to expand above a private memory size maximum; or the system manager must increase the job's private memory size maximum to accommodate the program. |
| ?Number is needed | A character string or variable name was used where a number was necessary. |
| ?1 or 2 dimensions only | Attempt was made to dimension a matrix to more than two dimensions. |
| ?ON statement needs GOTO | A statement beginning with ON does not contain a GOTO or GOSUB clause. |
| Please say HELLO | Message printed by the LOGIN system program. User not logged into the system has typed something other than a legal, logged-out command to the system. |
| ?Please use the RUN command | A transfer of control (as in a GOTO, GOSUB or IF-GOTO statement) cannot be performed from immediate mode. |
| ?PRINT-USING buffer overflow | Format specified contains a field too large to be manipulated by the PRINT-USING statement. |
| ?PRINT-USING format error | An error was made in the construction of the string used to supply the output format in a PRINT-USING statement. |
| ?Program lost-Sorry | A fatal system error has occurred which caused the user program to be lost. This error can indicate hardware problems or use of an improperly compiled program. Consult the system manager or the discussion of such errors in the *RSTS/E System Manager's Guide*. |

| Message Printed | Meaning |
|---|---|
| ?Redimensioned array | Usage of an array or matrix within the user program has caused BASIC-PLUS to redimension the array implicitly. |
| ?RESUME and no error | A RESUME statement was encountered where no error had occurred to cause a transfer into an error handling routine via the ON ERROR GOTO statement. |
| ?RETURN without GOSUB | RETURN statement encountered in user program without a previous GOSUB statement having been executed. |
| %SCALE factor interlock | An attempt was made to execute a program or source statement with the current scale factor. The program runs but the system uses the scale factor of the program in memory rather than the current scale factor. Use REPLACE and OLD or recompile the program to run with the current scale factor. (C) |
| ?Statement not found | Reference is made within the program to a line number which is not within the program. |
| Stop | STOP statement was executed. The user can usually continue program execution by typing CONT and the RETURN key. |
| ?String is needed | A number or variable name was used where a character string was necessary. |
| ?Syntax error | BASIC-PLUS statement was incorrectly formatted. |
| ?Too few arguments | The function has been called with a number of arguments not equal to the number defined for the function. |
| ?Too many arguments | A user-defined function may have up to five arguments. |
| ?Undefined function called | BASIC-PLUS interpreted some statement component as a function call for which there is no defined function (system or user). |
| ?What? | Command or immediate mode statement entered to BASIC-PLUS could not be processed. Illegal verb or improper format error most likely. |
| ?Wrong math package | Program was compiled on a system with either the 2-word or 4-word math package and an attempt is made to run the program on a system with the opposite math package. Recompile the program using the math package of the system on which it will be run. |

# APPENDIX D
# BASIC-PLUS CHARACTER SET

## D.1 BASIC-PLUS CHARACTER SET

User program statements are composed of individual characters. Allowable characters come from the following character set:

| | |
|---|---|
| A through Z | Space |
| 0 through 9 | Tab |

and the following special symbols and keys:

| Key | Use and Section in BASIC-PLUS Language Manual |
|---|---|
| $ | Used in specifying string variables (Section 5.1), or as the System Library File designator (*RSTS-11 System User's Guide*). |
| % | Used in specifying integer variables (Section 6.1). Also denotes account [1,4] (Section 9.1.1). |
| ' " | Used to delimit string constants, i.e., text strings (Section 5.1.). |
| ! | Begins comment part of a line (Section 3.1). Also denotes account [1,3] (Section 9.1.1). |
| : | Separates multiple statements on one line (Section 2.3.1). |
| \ | Separates multiple statements on one line as the colon (:) also does. |
| # | Denotes a device or file #name, or is used as an output format effector (Chapter 7 and Section 10.3.4). Also denotes account number using current project number with a programmer number of 0 (Section 9.1.1). |
| , | Output format effector and list terminator (Section 3.3). |
| ; | Output format effector (Section 3.3). |
| & | Denotes account [1,5] (Section 9.1.1). |
| @ | Denotes the assignable account (Section 9.1.1). |
| LINE FEED | When used at the end of a line, indicates that the current statement is continued on the next line (Section 2.3.2). |
| ( ) | Used to group arguments in an arithmetic expression (Section 2.5). Also may be used to group project-programmer number. |
| [ ] | Used to group project-programmer number. |

| Key | Use and Section in BASIC-PLUS Language Manual |
|---|---|
| <> | Used to delimit file protection codes. |
| + - * / | Arithmetic operators (Section 2.5.3). |
| = | Replacement operator (Section 3.2).<br>Logical equivalence operator (Section 2.5.4). |
| < | Logical "less than" operator (Section 2.5.4). |
| > | Logical "greater than" operator (Section 2.5.4). |
| == | Logical "approximately equal to" operator (Section 2.5.4). |

## D.2 ASCII CHARACTER CODES

| Decimal Value | ASCII Character | RSTS Usage | Decimal Value | ASCII Character | RSTS Usage | Decimal Value | ASCII Character | RSTS Usage |
|---|---|---|---|---|---|---|---|---|
| 0 | NUL | FILL character | 43 | + | | 86 | V | |
| 1 | SOH | | 44 | , | COMMA | 87 | W | |
| 2 | STX | | 45 | – | | 88 | X | |
| 3 | ETX | CTRL/C | 46 | . | | 89 | Y | |
| 4 | EOT | | 47 | / | | 90 | Z | |
| 5 | ENQ | | 48 | 0 | | 91 | [ | |
| 6 | ACK | | 49 | 1 | | 92 | \ | Backslash |
| 7 | BEL | BELL | 50 | 2 | | 93 | ] | |
| 8 | BS | | 51 | 3 | | 94 | ^ or ↑ | |
| 9 | HT | HORIZ. TAB | 52 | 4 | | 95 | _ or ← | |
| 10 | LF | LINE FEED | 53 | 5 | | 96 | ` | Grave accent |
| 11 | VT | VERT. TAB | 54 | 6 | | 97 | a | |
| 12 | FF | FORM FEED | 55 | 7 | | 98 | b | |
| 13 | CR | CAR. RET | 56 | 8 | | 99 | c | |
| 14 | SO | | 57 | 9 | | 100 | d | |
| 15 | SI | CTRL/Q | 58 | : | | 101 | e | |
| 16 | DLE | | 59 | ; | | 102 | f | |
| 17 | DC1 | | 60 | < | | 103 | g | |
| 18 | DC2 | | 61 | = | | 104 | h | |
| 19 | DC3 | | 62 | > | | 105 | i | |
| 20 | DC4 | | 63 | ? | | 106 | j | |
| 21 | NAK | CTRL/U | 64 | @ | | 107 | k | |
| 22 | SYN | | 65 | A | | 108 | l | |
| 23 | ETB | | 66 | B | | 109 | m | |
| 24 | CAN | | 67 | C | | 110 | n | |
| 25 | EM | | 68 | D | | 111 | o | |
| 26 | SUB | CTRL/Z | 69 | E | | 112 | p | |
| 27 | ESC | ESCAPE[1] | 70 | F | | 113 | q | |
| 28 | FS | | 71 | G | | 114 | r | |
| 29 | GS | | 72 | H | | 115 | s | |
| 30 | RS | | 73 | I | | 116 | t | |
| 31 | US | | 74 | J | | 117 | u | |
| 32 | SP | SPACE | 75 | K | | 118 | v | |
| 33 | ! | | 76 | L | | 119 | w | |
| 34 | ” | | 77 | M | | 120 | x | |
| 35 | # | | 78 | N | | 121 | y | |
| 36 | $ | | 79 | O | | 122 | z | |
| 37 | % | | 80 | P | | 123 | { | |
| 38 | & | | 81 | Q | | 124 | \| | Vertical Line |
| 39 | ’ | APOSTROPHE | 82 | R | | 125 | } | |
| 40 | ( | | 83 | S | | 126 | ~ | Tilde |
| 41 | ) | | 84 | T | | 127 | DEL | RUBOUT |
| 42 | * | | 85 | U | | | | |

[1] ALTMODE (ASCII 125) or PREFIX (ASCII 126) keys which appear on some terminals are translated internally into ESCAPE.

## D.3 CARD CODES

The RSTS card driver can be configured for one of three different punched card codes. These are: DEC029 codes,[1] DEC026 codes and 1401 (EBCDIC) codes. The RSTS-11 DEC029 and DEC026 codes are the same as the DOS-11 card codes. The particular set of codes used on the system is determined by the system manager. In all cases, the end-of-file (EOF) card must contain a 12-11-0-1 punch or a 12-11-0-1-6-7-8-9 punch in column 0.

## D.4 RADIX-50 CHARACTER SET

| Character | ASCII Octal Equivalent | Radix-50 Equivalent |
|---|---|---|
| space | 40 | 0 |
| A-Z | 101-132 | 1-32 |
| $ | 44 | 33 |
| . | 56 | 34 |
| unused | | 35 |
| 0-9 | 60-71 | 36-47 |

The maximum Radix-50 value is, thus,

$$47*50^2 + 47*50 + 47 = 174777$$

The following table provides a convenient means of translating between the ASCII character set and its Radix-50 equivalents. For example, given the ASCII string X2B, the Radix-50 equivalent is (arithmetic is performed in octal):

```
  X = 113000
  2 = 002400
  B = 000002
      ------
X2B = 115402
```

[1]The DEC029 code used by RSTS/E complies with the ANSI standard for punched card codes. IBM uses a card code which differs in three characters. If the DEC029 card code is configured on the system, a patch can be installed to change the decode table so that cards punched for use on IBM equipment can be read without misinterpretation. See the RSTS/E Release Notes for information on the codes affected.

| Character | $ASCII_{10}$ | DEC029 | DEC026 | 1401 |
|---|---|---|---|---|
| { | 123 | 12 0 | 12 0 | UNUSED |
| } | 125 | 11 0 | 11 0 | UNUSED |
| SPACE | 32 | NONE | NONE | NONE |
| ! | 33 | 12 8 7 | 12 8 7 | 11 0 |
| " | 34 | 8 7 | 0 8 5 | 0 8 2 |
| # | 35 | 8 3 | 0 8 6 | 8 3 |
| $ | 36 | 11 8 3 | 11 8 3 | 11 8 3 |
| % | 37 | 0 8 4 | 0 8 7 | 0 8 4 |
| & | 38 | 12 | 11 8 7 | 12 |
| ' | 39 | 8 5 | 8 6 | 12 8 4 |
| ( | 40 | 12 8 5 | 0 8 4 | 8 7 |
| ) | 41 | 11 8 5 | 12 8 4 | 0 8 7 |
| * | 42 | 11 8 4 | 11 8 4 | 11 8 4 |
| + | 43 | 12 8 6 | 12 | 0 8 5 |
| , | 44 | 0 8 3 | 0 8 3 | 0 8 3 |
| - | 45 | 11 | 11 | 11 |
| . | 46 | 12 8 3 | 12 8 3 | 12 8 3 |
| / | 47 | 0 1 | 0 1 | 0 1 |
| 0 | 48 | 0 | 0 | 0 |
| 1 | 49 | 1 | 1 | 1 |
| 2 | 50 | 2 | 2 | 2 |
| 3 | 51 | 3 | 3 | 3 |
| 4 | 52 | 4 | 4 | 4 |
| 5 | 53 | 5 | 5 | 5 |
| 6 | 54 | 6 | 6 | 6 |
| 7 | 55 | 7 | 7 | 7 |
| 8 | 56 | 8 | 8 | 8 |
| 9 | 57 | 9 | 9 | 9 |
| : | 58 | 8 2 | 11 8 2 | 8 5 |
| ; | 59 | 11 8 6 | 0 8 2 | 11 8 6 |
| < | 60 | 12 8 4 | 12 8 6 | 12 8 6 |
| = | 61 | 8 6 | 8 3 | 11 8 7 |
| > | 62 | 0 8 6 | 11 8 6 | 8 6 |
| ? | 63 | 0 8 7 | 12 8 2 | 12 0 |

(continued on next page)

EOF is a 12-11-0-1 punch or a 12-11-0-1-6-7-8-9 punch.

| **Character** | **ASCII$_{10}$** | **DEC029** | **DEC026** | **1401** |
|---|---|---|---|---|
| @ | 64 | 8 4 | 8 4 | 8 4 |
| A | 65 | 12 1 | 12 1 | 12 1 |
| B | 66 | 12 2 | 12 2 | 12 2 |
| C | 67 | 12 3 | 12 3 | 12 3 |
| D | 68 | 12 4 | 12 4 | 12 4 |
| E | 69 | 12 5 | 12 5 | 12 5 |
| F | 70 | 12 6 | 12 6 | 12 6 |
| G | 71 | 12 7 | 12 7 | 12 7 |
| H | 72 | 12 8 | 12 8 | 12 8 |
| I | 73 | 12 9 | 12 9 | 12 9 |
| J | 74 | 11 1 | 11 1 | 11 1 |
| K | 75 | 11 2 | 11 2 | 11 2 |
| L | 76 | 11 3 | 11 3 | 11 3 |
| M | 77 | 11 4 | 11 4 | 11 4 |
| N | 78 | 11 5 | 11 5 | 11 5 |
| O | 79 | 11 6 | 11 6 | 11 6 |
| P | 80 | 11 7 | 11 7 | 11 7 |
| Q | 81 | 11 8 | 11 8 | 11 8 |
| R | 82 | 11 9 | 11 9 | 11 9 |
| S | 83 | 0 2 | 0 2 | 0 2 |
| T | 84 | 0 3 | 0 3 | 0 3 |
| U | 85 | 0 4 | 0 4 | 0 4 |
| V | 86 | 0 5 | 0 5 | 0 5 |
| W | 87 | 0 6 | 0 6 | 0 6 |
| X | 88 | 0 7 | 0 7 | 0 7 |
| Y | 89 | 0 8 | 0 8 | 0 8 |
| Z | 90 | 0 9 | 0 9 | 0 9 |
| [ | 91 | 12 8 2 | 11 8 5 | 12 8 5 |
| \ | 92 | 0 8 2 | 8 7 | 0 8 6 |
| ] | 93 | 11 8 2 | 12 8 5 | 11 8 5 |
| ↑ or ^ | 94 | 11 8 7 | 8 5 | UNUSED |
| ← or _ | 95 | 0 8 5 | 8 2 | 12 8 7 |

EOF is a 12-11-0-1 punch or a 12-11-0-1-6-7-8-9 punch.

**Radix-50 Character/Position Table**

| Single Character or First Character | | Second Character | | Third Character | |
|---|---|---|---|---|---|
| A | 003100 | A | 000050 | A | 000001 |
| B | 006200 | B | 000120 | B | 000002 |
| C | 011300 | C | 000170 | C | 000003 |
| D | 014400 | D | 000240 | D | 000004 |
| E | 017500 | E | 000310 | E | 000005 |
| F | 022600 | F | 000360 | F | 000006 |
| G | 025700 | G | 000430 | G | 000007 |
| H | 031000 | H | 000500 | H | 000010 |
| I | 034100 | I | 000550 | I | 000011 |
| J | 037200 | J | 000620 | J | 000012 |
| K | 042300 | K | 000670 | K | 000013 |
| L | 045400 | L | 000740 | L | 000014 |
| M | 050500 | M | 001010 | M | 000015 |
| N | 053600 | N | 001060 | N | 000016 |
| O | 056700 | O | 001130 | O | 000017 |
| P | 062000 | P | 001200 | P | 000020 |
| Q | 065100 | Q | 001250 | Q | 000021 |
| R | 070200 | R | 001320 | R | 000022 |
| S | 073300 | S | 001370 | S | 000023 |
| T | 076400 | T | 001440 | T | 000024 |
| U | 101500 | U | 001510 | U | 000025 |
| V | 104600 | V | 001560 | V | 000026 |
| W | 107700 | W | 001630 | W | 000027 |
| X | 113000 | X | 001700 | X | 000030 |
| Y | 116100 | Y | 001750 | Y | 000031 |
| Z | 121200 | Z | 002020 | Z | 000032 |
| $ | 124300 | $ | 002070 | $ | 000033 |
| . | 127400 | . | 002140 | . | 000034 |
| unused | 132500 | unused | 002210 | unused | 000035 |
| 0 | 135600 | 0 | 002260 | 0 | 000036 |
| 1 | 140700 | 1 | 002330 | 1 | 000037 |
| 2 | 144000 | 2 | 002400 | 2 | 000040 |
| 3 | 147100 | 3 | 002450 | 3 | 000041 |
| 4 | 152200 | 4 | 002520 | 4 | 000042 |
| 5 | 155300 | 5 | 002570 | 5 | 000043 |
| 6 | 160400 | 6 | 002640 | 6 | 000044 |
| 7 | 163500 | 7 | 002710 | 7 | 000045 |
| 8 | 166600 | 8 | 002760 | 8 | 000046 |
| 9 | 171700 | 9 | 003030 | 9 | 000047 |

# INDEX

# INDEX (Cont.)

# INDEX (Cont.)

# INDEX (Cont.)

# INDEX (Cont.)

# INDEX (Cont.)

# INDEX (Cont.)

# INDEX (Cont.)

# INDEX (Cont.)

## READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

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
- ☐ Non-programmer interested in computer concepts and capabilities

Name ________________ Date ________________

Organization ________________

Street ________________

City ________________ State ________ Zip Code ________
or
Country

Please cut along this line.

--------------------------------------------------------- **Fold Here** ---------------------------------------------------------

--------------------------------------------- **Do Not Tear - Fold Here and Staple** ---------------------------------------------