RSTS/E Programming Manual

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

digital equipment corporation · maynard. massachusetts

First Printing, July 1975

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM		

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

PREFACE

This manual describes RSTS/E special programming techniques, many of which are not available prior to Version 6 (RSTS/E) systems. Included in this manual are instructions and helpful hints for programming devices that RSTS/E supports. These techniques permit the programmer full control over input and output operations on specific devices; for this reason they are called device dependent operations.

System accounts and the concept of privileged users are also discussed. Chapter 7, as it appeared originally in the RSTS/E System Manager's Guide, discusses SYS functions, both for privileged and for non-privileged users.

The material covered in this manual assumes familiarity with the BASIC-PLUS Language Manual and the RSTS-11 System User's Guide, in addition to a thorough knowledge of the BASIC-PLUS language. For this reason, Senior Programmers and Application Programmers will be most interested in this manual.

For information on all of the current manuals pertaining to RSTS/E operation, consult the RSTS/E Documentation Directory.

CHAPTER 1

DISKS

1.1 SYSTEM ACCOUNTS

RSTS/E systems have three system accounts which are essential to the operation of the system. The Master File Directory (MFD) account [1,1] is used on the system device and other disk devices in the system to control access. It is described in Section 1.1.1. The system library account [1,2] is used by the RSTS/E system to manage a library of generally available and restricted use system programs and message and control files. It is described in Section 1.1.2. System account [0,1] contains RSTS/E Monitor files and routines which are critical to the operation of the system. System account [0,1] is described in Section 1.1.3.

1.1.1 MFD Account [1,1] On The System Device

Access to the RSTS/E system is controlled by the use of project-programmer numbers and passwords. The system manager, operating under his privileged account, creates a new account by using the system program REACT. The project-programmer number and password of the new account are given, along with other information, to allow a user access to system facilities.

When a new account is created, the new account information is stored on the system device under the MFD account [1,1]. The password is stored in packed Radix-50 format. When the new user first creates a file, an area is created on the system device which is related directly to the user's account (project-programmer number). This area is called the User File Directory or UFD. The UFD contains information concerning the files created under that account number.

The account [1,1] contains a catalog of information of all User File Directories on the system and is called the Master File Directory or MFD. When a user attempts to gain access to the RSTS/E system by giving his account and password, the system program LOGIN runs automatically. LOGIN checks the MFD on the system device to determine whether the account number and password given compare with one stored in the MFD. If so, the user is allowed access to the system.

Other account information is stored in the MFD for each account in the system. This information is summarized in Table 1-1.

The account information in the MFD is accessed by various system programs. The LOGIN program has already been mentioned. The MONEY system program references the accumulated system accounting information. The system manager uses the MONEY system program to print and reset this accounting data. The disk storage information is referenced by the LOGOUT system program. The system manager can change the disk quota by use of the UTILTY system program.

Table 1-1
Account Information Stored in the MFD on the System Device

Type	Description	Explanation
Identification	Project-programmer number (account)	Refer to Chapter 9 of the <u>BASIC-PLUS Language Manual</u>.
	Password	6 characters stored within 2 words in Radix-50 format.
Accumulated Usage	Central Processor Unit (CPU) time (Run Time)	Processor time the account has used to date in tenths of a second.
	Connect time (login time)	Number of minutes the user has been connected to the system via a terminal or remote line.
	Kilo-core-ticks (KCT's)	Core usage factor. One KCT is the usage of 1K words of memory for one tenth of a second.
	Device time	Number of minutes of peripheral device time the account has used.
Disk Storage	Quota	Number of 256-word blocks the user is allowed to retain at log out time.

Using System Function calls, the system manager can write programs which access the information in the MFD. See the description of the system function calls in Chapter 7.

1.1.1.1 MFD On Non-System Disks

The system disk exists in what is called the public structure. Additional disk devices can be added to the public structure or can be added as private packs. Each disk that is added to the system also contains its own MFD. The MFD of each additional disk is created when the system manager uses the initialization option DSKINT. The MFD of a disk contains account and storage information for that device only.

The MFD on public disks is treated differently from the MFD on private disks. The RSTS/E system allocates space for a user's file on the disk in the public structure that has the most free space. If the user's account is not in the MFD of the disk with the most free space, his account number is added dynamically into the MFD of that disk and a UFD is created for the user on that disk. If a user desires to create a file on a private pack he cannot do so if his account number does not already exist in the MFD of that device. The system manager or a privileged user grants access to a private disk by entering the account information on the desired disk with the REACT system program.

The MFD on a disk device contains that disk's pack label. The pack label information consists of pack cluster size, pack status (private or public), and pack identification (id). The pack id is the 6-character name in Radix-50 format given at the time the disk was initialized via the initialization option DSKINT. The pack id is utilized whenever the disk is logically mounted via the system program UTILTY or via INIT. (A distinction must be made between physical mounting and logical mounting. The disk is mounted physically by making the hardware ready to use the disk. The disk must be mounted logically by the system program UTILTY or UMOUNT or from commands in the START.CTL or CRASH.CTL files by the INIT system program. The disk must also be logically dismounted <u>before</u> it is physically dismounted to avoid corruption of the file structures.)

1.1.2 System Library Account [1,2]

The system library account [1,2] is created on the system device during the DSKINT operation of building the system disk. During the library build procedures, the system manager creates the contents of the system library account [1,2]. This section describes briefly the contents of account [1,2]. The entire content of the account is tabulated in the <u>RSTS/E System Manager's Guide</u>.

The system library catalogs system programs which are available to general users and to privileged users. It also contains text and control files used by system programs.

For operational purposes, the system library is accessed automatically during normal system start-up. As a result of specifying the START option when RSTS/E is bootstrapped into memory, the console keyboard is logged in automatically under account [1,2]. At this time, the commands in the file START.CTL or in another command file, stored under account [1,2], are executed by the INIT system program, which is also stored under account [1,2]. Unless the system manager desires to modify or add to the contents of the system library after the system disk has been built, he should not log into the system under account [1,2].

For automatic restart purposes, the system library file CRASH.CTL is used by the INIT system program when recovering from system crashes. When automatic restart mode is entered, the RSTS/E system performs actions similar to those described above for normal start-up, except that the commands in the CRASH.CTL file are executed.

Two files, HELP.TXT and NOTICE.TXT, are provided to supply information to the user. The HELP.TXT file is printed automatically by LOGIN if a user types HELP at a logged-out terminal. The NOTICE.TXT file is printed automatically by LOGIN after a user has typed a valid account number and password during the log-in procedure. The NOTICE.TXT file provides a means by which the system manager can communicate up-to-date installation information to users. Another system library information file, PIP.TXT, is printed out when the user types /HELP while running the related system program, PIP.

The file GRIPE.TXT is created dynamically when a user runs the GRIPE system program. The system manager has read and delete access to the GRIPE.TXT file by means of the GRIPE system program.

The file QUEUE.SYS is the system queue file and contains user requests created by the QUE program. The system queue manager program QUEMAN creates the file and initializes it. All spooling programs running on the system receive their requests from the QUEUE.SYS file through the QUEMAN program.

The ERRLOG.FIL file contains a history of hardware errors logged on the RSTS/E system. The ERRCPY program transfers the error data from monitor storage to the file ERRLOG.FIL. The system manager gains information on its contents and deletes its contents by running the ERRDIS program.

1.1.3 System Account [0,1] on the System Device

The system account [0,1], like the system library account, is created on the system device during the DSKINT operation of building the system disk. Account [0,1] has a password identical to the pack id, and it contains name and retrieval information for system files. Account [0,1] is used solely by the RSTS/E Monitor to execute various operational and control actions. In this respect, it is a direct access account for use by the RSTS/E Monitor, and no user need be concerned with referencing its contents. This section describes the contents of the account, for a better understanding of the internal operations of the system.

The system manager creates and organizes the files stored under account [0,1] when he rebuilds the system disk using the REFRESH option. The possible contents of account [0,1] are described in the following paragraphs.

1.1.3.1 Allocating Disk Storage Space - the file SATT.SYS is the mechanism by which RSTS/E controls the allocation and deallocation of storage space for the disk. The file maps the entire space on the disk in a bit map called a SAT (Storage Allocation Table). Each bit in a SAT represents either allocated or unallocated space. The system sets a bit in the SAT to 1 when that space is allocated for any purpose.

The system allocates storage space in terms of clusters. Each bit in the SAT represents one cluster of disk space. A cluster is a fixed number of contiguous 256-word blocks of storage on the disk. The cluster size, or cluster factor, defines how many contiguous 256-word blocks are contained in the cluster.

Cluster sizes in RSTS/E are defined for disks, directories, and files. Table 1-2 presents the types of clusters and related information.

Table 1-2
Valid Cluster Size Ranges

Cluster Size	Minimum Size	Maximum Size	When Defined

Pack (for non-system disk, public and private)	1(RF11/RK05)
	2(RP02/RP03)
	4(RP04)
		16(10)	At initialization time via DSKINT option (stored in MFD)

Directory (both for MFD and UFD)	Pack Cluster Size	16(10)	At creation of the directory via either the DSKINT initialization option, REACT, or SYS system function.
File	Pack Cluster Size	256(10)	At creation of the file via either an OPEN or OPEN FOR OUTPUT.

The system manager specifies the cluster size of the disk when he initializes it and gives a value in response to the PACK CLUSTER SIZE question. The pack cluster size defines the number of contiguous 256-word blocks a cluster comprises and, therefore, the extent of contiguous space represented by each bit in the SAT. A pack cluster size of 1 means that one 256-word block of storage is allocated for each bit set to 1. A pack cluster size of 2 means that two contiguous 256-word blocks are allocated for each bit set to 1. Allowable pack cluster sizes for an RF or an RK type disk are 1, 2, 4, 8, or 16; for an RP02/RP03 type disk, 2, 4, 8, or 16; and for an RP04 type disk, 4, 8 or 16.

The pack cluster size affects the efficiency of storage space allocation. A large size improves access time to system program and user files, but may waste disk space. For example, if the pack cluster size is 16, a one block file on the disk has allocated one cluster of 16 contiguous blocks. Fifteen blocks are wasted. A 15 block file also requires one cluster but only one block is wasted. Thus, the system manager must choose the pack cluster size which best fits the type of processing and the access requirements of the local installation.

A directory cluster size is defined for both a master file directory (MFD) and user file directory (UFD) and its minimum value is the pack cluster size. The system manager specifies the MFD cluster size when he initializes the disk; he specifies the UFD cluster size when he creates an account. A directory cluster size is equal to the pack cluster size or a power of 2 to a maximum of 16. Thus, for a pack cluster size of 2, the directory cluster size on that device can be 2, 4, 8, or 16. For a pack cluster size of 8, a directory cluster size on that device can be 8 or 16.

The directory cluster size limits the size to which a directory can expand. A directory, whether an MFD or a UFD, expands to catalog accounts or files but can occupy a maximum of seven clusters. For an MFD on the system disk, the cluster size determines how many accounts the system can handle. The following formula gives the number of user accounts, A, for each allowable MFD cluster size, MC.

$$\frac{(217 \times MC) - 1}{2} = A$$

The minimum number of accounts (A) is 108 for an MFD cluster size of 1 and the maximum is 1735 accounts for an MFD cluster size of 16.

The UFD cluster size determines how many files a user can create under one account. The following formula gives the number of user files, UF, for each allowable UFD cluster size, UC. (The formula assumes that all files are a minimum size between 1 and 7 clusters.)

$$\frac{(217 \times UC) - 1}{3} = UF$$

The minimum number of user files (UF) is 72 for a UFD cluster size of 1 and the maximum UF is 1157 for a UFD cluster size of 16.

1.1.3.2 System Overlay File and DECtape Directory - The files OVR.SYS and BUFF.SYS are crucial to RSTS/E system response time. Optimum efficiency is gained when these files reside on a fast access, fixed head disk. If the system disk is not a fixed head disk, the system manager can use the REFRESH option and cause a copy of the non-resident (overlay) code to be created, placed in the file OVR.SYS, and stored on the fast access disk, if available. On systems with only a moving head disk, it may be more efficient to create the file OVR.SYS and optimally position it.

DECtape processing is expedited by the use of BUFF.SYS. When a file on DECtape is opened, the directory of the DECtape is written to the file BUFF.SYS. The BUFF.SYS file requires three 256-word blocks for each DECtape drive on the system. Any updates to the DECtape directory which arise during processing cause the system to manipulate the copy in BUFF.SYS. This technique eliminates the need for continuous winding and rewinding of DECtape. The copy of the DECtape directory in BUFF.SYS is read back to the DECtape when the last file open on the DECtape unit is closed or any output file is closed. By use of the REFRESH option, the BUFF.SYS file can be placed on a fast access type disk or can be optimally positioned on the moving head system device.

1.1.3.3 System Operations File - The RSTS/E Core Image Library resides in the file RSTS.CIL on the system disk. This file contains all executable code for system operations, resident and non-resident, and optionally contains the stand-alone program ROLLIN.

1.1.3.4 Error Messages File - A copy of the BASIC-PLUS error messages from the CIL (RSTS.CIL) can be kept in the file ERR.SYS, which can be accessed and modified.

1.1.3.5 Saving Information After a Crash - The CRASH.SYS file can be created to retain a copy of valuable contents of memory at the time of a system crash.

1.1.4 System Account [0,1] on Non-System Disks

The system account [0,1] on non-system disks contains only a BADB.SYS file and a SATT.SYS file. The system account on non-system disks is created when the user initializes the disk.

1.2 PRIVILEGE

Certain accounts in the RSTS/E system have special capabilities outlined in this section. An account having these special capabilities is called a privileged account. (In the same way, when a user is assigned a privileged account, he is a privileged user.) The system manager, operating under his own privileged account, designates other users as privileged by assigning them an account with a project number 1 when he creates it with REACT. The system library account [1,2], discussed in a preceding section of this section of his chapter, is an example of a privileged account. The available privileged account numbers are [1,3] through [1,254]. Privileged accounts have the following special capabilities:

a. Unlimited file access. See Section 1.2.1.

b. Creation and modification of privileged programs. See Section 1.2.2.

c. Use of privileged aspects of system programs. See Section 1.2.3.

d. Use of privileged SYS functions and the PEEK function. See Chapter 7.

The above listed capabilities are described in the remaining sections of this chapter and in Chapter 7. It must be emphasized that there is no fail safe and that no error messages are generated if the use of such special capabilities is to result in destruction of the system. For this reason, it is suggested that the system manager assign privileged accounts sparingly. It is recommended that the system manager create additional non-privileged accounts for himself and perform most of his functions under them. The system manager should use a privileged account only when necessary.

1.2.1 Unlimited File Access

No file in the RSTS/E system can be protected against the user of a privileged account. A privileged user can create and delete files under any account number and can access files on LOCKED disks. Under such circumstances, no protection violation occurs.

1.2.2 Creation And Modification Of Privileged Programs

A program is privileged when it has a protection code of <192> or greater. Protection code <192> means that the privileged protection <128> and the compiled file protection <64> are set. Both protection code values must be set for a program to have privileged status. The system manager or privileged user designates a program as privileged by assigning to its compiled form a protection code of <192> or greater with the NAME-AS command. For example:

```
NAME "FILE.BAC" AS "FILE.BAC <232>"
```

designates the compiled program FILE as privileged with write protection against the owner's group and all others not in the owner's group. Refer to Chapter 9 of the BASIC-PLUS Language Manual for the definition of protection codes.

If a program is designated privileged, any user can run the program with privileged program status (provided he has READ access to the file). The privileged program status exists only for the duration of the program run. Privileged program status means that system operations normally reserved to a user of a privileged account can be executed while running under a non-privileged account. If the user running a privileged program interrupts execution of the program by typing CTRL/C, the program loses its privileged status.

The ability to designate a program as privileged allows the system manager to extend use of privileged functions to non-privileged users. For example, the program TTYSET allows a user to change characteristics of his terminal. Such an action is a privileged system function executable only by owners of privileged accounts. With the privileged program status, execution of the function by the owner of a non-privileged account does not cause the normal program trap.

The same TTYSET program additionally allows a privileged user to change characteristics of a terminal other than his own. A check is built into the program to ensure that a user attempting to change the characteristics of a terminal other than his own is indeed a privileged user. As a result, the execution of privileged functions is made available to the non-privileged user but some privileged features are restricted. The system manager likewise can control the use of privileged operations.

1.2.3 Use Of Privileged Features Of System Programs

The owner of a privileged account can execute the privileged features of system programs. Since the list of privileged features is lengthy, an entire chapter in the RSTS/E System Manager's Guide is devoted to explaining them. Certain programs, such as TTYSET, SYSTAT, and MONEY, are privileged but contain features helpful to the general user. These programs are therefore described in the RSTS-11 System User's Guide for the non-privileged user.

Two of the programs, TTYSET and MONEY, have privileged features about which the non-privileged user is not informed in the User's Guide. TTYSET is discussed in Chapter 4 of the RSTS/E System Manager's Guide. MONEY is discussed in Chapter 7 of this manual. Another program, SYSTAT, is discussed in Chapter 4 of the Manager's Guide and Chapter 7 of this manual, although it contains few explicit privileged features.

1.3 NON-FILE STRUCTURED DISK OPERATION

In non-file structured processing of disks, the user program can access specific physical blocks on the disk. The unit on which a disk is being processed in non-file structured mode may or may not be logically mounted. To initiate non-file structured processing, the program gives only a device designator in the OPEN statement. Of the three conventional forms of the OPEN statement, OPEN FOR INPUT, OPEN, and OPEN FOR OUTPUT, only two are valid. The following two sample statements,

```
100  OPEN "DK1:" FOR INPUT AS FILE 1%
```

and

```
100  OPEN "DK1:" AS FILE 1%
```

are equivalent statements because both reading and writing of physical blocks on the device are permitted. A third sample statement:

```
100  OPEN "DK1:" FOR OUTPUT AS FILE 3%
```

is invalid since it attempts to create a file.

Subsequent to opening a disk device for non-file structured processing, GET and PUT statements can retrieve and write specific physical blocks on the device by means of the RECORD option. The record number is interpreted as a device cluster number. The device cluster size is the same as the minimum pack cluster size for that device type. The default buffer size is the device cluster size multiplied by 512 bytes. The following example reads the last 2 blocks of an RP03.

```
100  OPEN "DP1:" AS FILE 1%:GET #1%, RECORD 30000% + 9999%
```

Device clusters are numbered starting at 0. Device cluster 0 can be accessed only immediately following an OPEN statement. The GET or PUT statement used to access device cluster 0 must either include RECORD 0 or omit the RECORD option. Once device cluster 0 has been accessed, omitting the RECORD option or specifying RECORD 0 in a GET or PUT statement accesses the next sequential device cluster. The only way to access device cluster 0 again is to close the device and reopen it for non-file structured access.

NOTE

In file structured disk operations, file blocks are numbered form 1 to the last block of the file. A GET or PUT statement that omits the RECORD option or specifies RECORD 0 accesses the next sequential block in the file, since RECORD 0 of a file does not exist.

A user can access a disk in non-file structured mode without logically mounting it. The disk can be accessed by a user (either privileged or non-privileged) for reading and writing, once the disk is physically inserted in the disk drive.

A non-privileged user cannot access a disk in non-file structured mode if some other user is accessing it. Attemping to access the disk under these condition results in a PROTECTION VIOLATION (ERR = 10) error for a non-privileged user. A privileged user can read the disk in non-file structured mode, regardless of how many users are accessing the disk; but the privileged user cannot write on the disk in non-file structured mode if it is being accessed by another user.

Once the disk is logically mounted, a privileged user is granted only read privileges in non-file structured mode. A non-privileged user is granted neither write nor read access in non-file structured mode, when the disk is logically mounted.

1.4 UPDATE OPTION FOR DISK FILES

In the description of disk files up to this point, the concept of simultaneous user access to a single file has been largely ignored. The system permits several users to read from a single file simultaneously, but a problem arises when multiple users attempt to write onto a single file simultaneously. Two users could conceivably try to write the same record of the file, resulting in a loss of data. To avoid this conflict, the system normally permits only one user at a time to have write privileges on any given file. Thus, a user may fail to obtain write privileges even if the file is not protected against writing. If this occurs the user must close the file and reopen it at a later time, after the other user has finished with the file and closed it.

In certain applications (for example, sales order-entry applications) it might be normal for multiple users to be updating a single master file. In such cases it is not satisfactory to constantly close and reopen the file to obtain write privileges; this is a time-consuming operation. For this reason a special UPDATE option is available that permits multiple users to have write access to a file while guarding against simultaneous writing of a single record.

To indicate that a file is being opened for UPDATE, the MODE 1% specification is used when the file is opened. For example:

```
100  OPEN "MASTER" AS FILE 1%, MODE 1%
```

when used with a disk file indicates that the file is opened for UPDATE(1).

In this case the program is granted write privileges

----

(1)The RECORDSIZE option may not be used on files that are opened for UPDATE because only one block (512 bytes) can be locked on a given channel at a time by a user.

unless such access is specifically prohibited by the protection code of the file.

A file cannot be simultaneously open for UPDATE by one user and open in normal (non-UPDATE) mode by another user. Attempting to open a file for UPDATE if it is already open in normal mode, or attempting to open a file in normal mode if it is already open for UPDATE, results in a PROTECTION VIOLATION error.

Once a file has been opened for UPDATE, any read operation of a specific record puts that block in a special "locked" state. This means that no other user is permitted to read or write that record until it is released (or unlocked) by the program that locked it. Attempting to read or write a record that another user has locked results in a DISK BLOCK IS INTERLOCKED (ERR = 19) error which can be trapped with an ON ERROR GOTO command. There are five ways for a program to unlock the record:

1. The next write operation on the file unlocks the record.

2. Executing an UNLOCK statement. This statement has the form:

   ```
   line number UNLOCK # <expr>
   ```

   where <expr> is the internal channel number of the file that is opened for UPDATE.

3. Any error encountered while accessing the file unlocks the RECORD.

4. Reading another record unlocks the currently locked record. Any read operation locks the record just retrieved.

5. Executing a CLOSE statement on the file unlocks the record.

To illustrate UPDATE, consider a simple inventory application where operators on several terminals can access one file to enter a part number and order quantities. Assume that the file is sequenced so that each part number actually corresponds to the record number that contains information about the part, and that the first four characters of the 512-byte record contain the quantity available as a (2-word) floating-point number. For this example, the remaining 508 characters are ignored. A program to update the quantity available is as follows:

```
100 ON ERROR GOTO 1000                      !FIND OUT ABOUT ERRORS
200 OPEN "INVENT.ORY" AS FILE 1, MODE 1     !OPEN FILE IN UPDATE MODE
300 FIELD #1, 4 AS C$                       !C$ IS QTY IN FILE
400 INPUT "PART NUMBER";N;"QUANTITY";Q      !GET PART # AND QTY
500 GET #1, RECORD N                        !READ APPROPRIATE RECORD
600 X=CVT$F(C$)-Q                           !COMPUTE QTY REMAINING
700 IF X>=0 THEN 800                        !ENOUGH ON HAND?
710 UNLOCK #1                               !PERMIT OTHER ACCESSES
720 PRINT "ONLY" CVT$F(C$) "ITEMS LEFT"     !REPORT STOCK LEVEL
730 GOTO 400                                !REVISE ORDER?
800 LSET C$=CVTF$(X)                        !STORE NEW QTY ON HAND
850 PUT #1, RECORD N                        !REWRITE INTO FILE
900 GOTO 400                                !NEXT TRANSACTION
1000 IF ERR <>19 THEN ON ERROR GOTO 0       !IGNORE NON-INTERLOCK
                                            !ERRORS
1100 PRINT "WAITING"                        !LET HIM KNOW WE'RE HERE
1200 SLEEP 5                                !WAIT FOR CURRENT ACCESS
1300 RESUME 500                             !TRY AGAIN
1400 END
```

A privileged user can write into a UFD/MFD only by using MODE 16384% in the OPEN statement. For example,

```
OPEN "DK1:[5,10]" AS FILE 2%, MODE 16384%
```

allows a privileged user to read and write into the UFD of account [5,10].

Specifying MODE 5% (1%+4%) in an OPEN statement requires a record to be already locked (via a previous GET statement) for a PUT statement to be executed. If the record is not locked, attempting to execute PUT results in a PROTECTION VIOLATION (ERR = 10) error.

1.5 APPENDING DATA TO A DISK FILE

To write data to a new block following the current end of file in a disk file, specify the MODE 2% option in the OPEN statement. Mode 2% should be used only with Record I/O files. For example,

```
100  OPEN "DATA" AS FILE 1%, MODE 2%
```

The system opens the file DATA under the current account on the system disk. The next output operation creates a new block and appends it to those currently allocated to the file. Any fill characters in the previous last block of the file remain when the system appends the new last block. Do not use the OPEN FOR OUTPUT form of the OPEN statement, as it deletes the existing file.

1.6 FLOPPY DISKS

The RX11/RX01 Floppy Disk can be used only as a non-file structured device. The device name for the floppy disk is DX. Unit numbers for the floppy disk drive start at 0. The current floppy disk implementation allows only a single .BAS file to be stored on floppy disk unit 1, for example, with the SAVE command:

```
          SAVE DX1:
```

It can be read from the disk or run by the following respective commands:

```
          OLD DX1:

               and

          RUN DX1:
```

A floppy disk is divided into 77 tracks (numbered 0 through 76), each of which consists of 26 sectors (numbered 1 through 26). Consequently, there are 2002 (77 x 26) 128-byte records (numbered 0 through 2001) on each disk.

A floppy disk can be opened and accessed in either of two modes: sector mode or block mode.

1.6.1 Sector Mode

In sector mode the buffer size is 128 bytes. Open the floppy disk on unit 3 in sector mode with the following statement:

```
          OPEN "DX3:" AS FILE 1%, MODE 16384%
```

When the GET and PUT statements are used, track and sector numbers can be calculated once the RECORD number is known. If the desired record number is specified as N (any number from 0 through 2001), the track and sector accessed can be determined as follows:

```
          TRACK = INT (N/26)

          SECTOR = N - INT(N/26)*26 + 1
```

A GET statement reads a 128-byte record from the disk. The RECORD option, if present, defines a specific record on the disk. If the RECORD option is omitted or RECORD 0% is included, the next sequential record is read.

```
          100 GET #1%, RECORD N%
```

In the above statement, N% is the record number and can be any number from 1 through 2001. (Record 0 can be accessed only by the first GET statement after the file has been opened.)

If -32768% (formed by 32767% + 1%) is included in the RECORD option (e.g., RECORD N%-32767%-1%), sectors are interleaved according to the algorithm discussed in Section 1.6.2.

A PUT statement writes a 128-byte record on the disk.

```
          200 PUT #1%, RECORD N%, COUNT C%
```

In the above statement, N% is the record number. The RECORD option can also include -32768% for interleaving (see Section 1.6.2) and 16384% to write a Deleted Data Mark with each of the records (see Section 1.6.3). C% must be a multiple of 128.

1.6.2 Block Mode

In block mode the buffer size is 512 bytes, equivalent to four 128-byte records. The four sectors are interleaved according to the following algorithm where N is the value specified in RECORD:

```
          TEMP1 = INT(N/26)

          TEMP2 = N - INT(N/26)*26

          TEMP2 = TEMP2 * 2

          TEMP2 = TEMP2+1 IF TEMP2 >=26

          TEMP2 = TEMP2 + 6*TEMP1

          TRACK = TEMP1 + 1

          SECTOR = TEMP2 - INT(TEMP2/26)*26 + 1
```

The above interleaving algorithm is standard for other PDP-11 operating systems for the floppy disk (e.g., RSX-11M, RT11). Track 0 is unavailable in order to reserve its use for IBM-compatible labels.

The statement shown below opens the floppy disk on unit 3 in block mode on I/O channel 1.

```
          OPEN "DX3:" AS FILE 1%
```

A GET statement reads a 512-byte block from the disk. The RECORD option, if present, defines a specified sector starting point for the read. If the RECORD option is omitted or RECORD 0% is included, the next sequential block is read.

```
        100 GET #1%, RECORD N%
```

In the above statement, N% is the number of the sector at which the block begins. It can be any number from 1 through 493. (The first block on the disk can be accessed only by the first GET statement after the device is opened.)

A PUT statement writes a 512-byte block on the disk.

```
        200 PUT #1%, RECORD N%, COUNT C%
```

In the above statement, N% is the number of the sector at which the block begins. It can also include 16384% to write a Deleted Data Mark with each of the sectors (see Section 1.6.3). C% must be a multiple of 128.

Block mode operations can be performed in sector mode by opening the disk with this statement:

```
        OPEN "DX3:" AS FILE 1%, RECORDSIZE 512%, MODE 16384%
```

and using the GET (or PUT) statement as follows:

```
        GET #1%, RECORD N%*4% + 32767% + 1%
```

In the above statement, 32767% + 1% specifies sector interleaving and N%*4% defines 512-byte blocks at 4-sector intervals.

1.6.3 Deleted Data Marks

Each sector of a floppy disk contains a bit called the Deleted Data Mark, in addition to its 128 bytes of data. When an INPUT or GET operation from the disk encounters a Deleted Data Mark, the DATA FORMAT ERROR (ERR = 50) occurs.

In the case of a GET operation, the contents of the buffer are valid even if this error occurs. So it is possible to examine the contents of the record containing the Deleted Data Mark. When the recordsize specified is larger than 128 bytes (e.g., block mode operation), the last 128 bytes read into the buffer are the data that had the Deleted Data Mark. The RECOUNT variable reflects the amount of data read up to and including this mark.

1.7 DEFAULT BUFFER SIZES

The devices shown below have the following default characteristics:

Device	Device Clustersize	Default Buffer Size (non-file structured) in Bytes	Maximum Record No.
DF0:	1	512	RECORD (#platters*1024)-1
DK0: - DK7:	1	512	RECORD 4799
DP0: - DP7:	2	1024	RECORD 39999
DB0: - DB7:	4	2048	RECORD 41799
DX0: - DX7:	1	128	RECORD 2001

CHAPTER 2

MAGTAPE

Magtape I/O is processed under RSTS-11 in one of two forms: file structured magtape and non-file structured magtape. In addition to this distinction, RSTS/E supports magtape files both in DOS-11 and in ANSI formats. (See Appendix A for detailed information on DOS vs. ANSI magtape file labels.)

<u>NOTE</u>

> For accurate results, all files stored on a given magtape should be written in the same format (DOS-11 or ANSI). Mixing file types on one tape can result in illogical interpretations.

A user program controls file structured magtape operations by the MODE specification in the OPEN statement. MODE determines how the system searches for a file with each of the three forms of the OPEN statement described in Section 9.2 of the <u>BASIC-PLUS Language Manual</u>. It also controls the system action when the file is subsequently closed.

The following sections describe magtape operations in some detail. Sections 2.1 through 2.3 describe the use of MODE with each of the forms of the OPEN statement. Section 2.4 describes magtape operation on a CLOSE statement. The remainder of this chapter discusses non-file structured magtape operations, the MAGTAPE function, and error handling techniques.

2.1 THE FILE STRUCTURED MAGTAPE OPEN FOR INPUT

Once a filename is specified in the OPEN statement, that magtape file is opened for file structured processing. For example:

```
          100  OPEN "MT0:ABC" FOR INPUT AS FILE N%, MODE M%
```

The OPEN FOR INPUT statement searches for a specified file on a designated magtape unit. A BASIC-PLUS program executes the statement so that it can subsequently perform input from the file. (Unlike disk operation, OPEN FOR INPUT on magtape permits read access only.)

In the above example, the system associates magtape unit 0 with the channel designated by N% and searches for file ABC under the current account according to the value of M% in the MODE specification.

The meanings shown below can be attached to MODE values used in an OPEN FOR INPUT statement. The MODE value can be the sum of any combination of these single values, as long as they are not mutually exclusive.

MODE	Meaning
0%	Read record at current tape position.
2%	Do not rewind tape when searching for specified file.
32%	Rewind tape before searching for specified file.
64%	Rewind tape upon executing a CLOSE.
16384%	Search for a DOS formatted label.
24576%	Search for an ANSI formatted label.

If the file is found, it is open for read access only. A GET statement subsequently executed on channel N% makes a block of the file available to the program in the channel's buffer. Since the file is open solely for input, a PUT statement subsequently executed on the channel generates the PROTECTION VIOLATION error (ERR = 10). If the system detects a logical end of tape before finding a file, the CAN'T FIND FILE OR ACCOUNT error (ERR = 5) occurs.

To open a file stored on the magtape under an account other than the current account, simply supply the proper project-programmer number in the OPEN statement. Since the system does not check protection codes for files on magtape, any user can access a magtape file.

2.1.1 Searching For A Record

Omitting the MODE specification or using a MODE 0% specification reads the record at the current position of the tape. If the system finds a label record and its file name and account match those of the file specified in the OPEN statement, the system generates the NAME OR ACCOUNT NOW EXISTS error (ERR = 16). No match causes the system to rewind the tape and check successive label records until the record is found or the logical end of tape is detected. The system does not rewind the tape when the program executes a CLOSE statement on channel N%.

2.1.2 Rewinding The Tape

As mentioned before, MODE 0% reads the tape from its current position. If the filename specified in the OPEN statement does not match the record label, the system automatically rewinds the tape to the first label record and begins reading labels file by file.

To override this automatic rewind feature, include the MODE value 2% in the OPEN statement. In this case, the system reads the tape from its current position and, if no match occurs, continues reading label records from that position forward until either the file is found or until the logical end of tape is detected. The system normally does not rewind the tape when it performs a CLOSE operation.

MODE 32% rewinds the tape to the first label record before reading any label. Once again, no match causes the system to check successive label records until the file is found or until the logical end of tape is detected. The system does not rewind the tape when it performs the CLOSE operation on channel N%. Finally, including MODE value 64% rewinds the tape when a CLOSE statement is executed on channel N%.

2.1.3 DOS and ANSI Format Labels

Including MODE 16384% in the OPEN FOR INPUT statement searches for the magtape file whose filename is specified. In addition, the file must be written in DOS format (i.e., preceded by a DOS label) for the search to be considered successful.

When the user omits the MODE value 16384%, the system assumes record labels on the tape, either DOS or ANSI, are in the default format specified by the system manager when he starts time-sharing operations. The default format can also be defined with an ASSIGN statement.

Including MODE 24576% in the OPEN FOR INPUT statement searches for the magtape file whose filename is specified. In this case, the file must be written in ANSI format (i.e., preceded by an ANSI label) for the search to be considered successful.

The system reads the tape at its current position. If the file is not found, it rewinds the tape and reads label by label until it finds the correct file. If the logical end of tape is detected, the CAN'T FIND FILE OR ACCOUNT error (ERR = 5) occurs.

Once again, omitting the MODE value 24576% in the OPEN FOR INPUT statement assumes record labels are in the format originally specified by the system manager.

2.1.4 Example of OPEN FOR INPUT Statement

The values for MODE discussed above can be combined in any combination as long as they are not mutually exclusive. (For example, MODE 16384% is incompatible with MODE 24576%, so MODE 16384%+24576% causes illogical results.)

Consider the following line:

```
10  OPEN "MT1:MARKIE" FOR INPUT AS FILE 3%, MODE 24772%
```

This line opens the file MARKIE on magtape unit 1 and associate it with channel 3%. MODE 24772% is actually the sum of MODE 32% + 64% + 24576%.

When the above line is executed, the system rewinds the tape to the first record label (MODE 32%) and begins to read successive record labels until the file is found or the logical end of tape is reached. The search is successful only if the system finds the file MARKIE, written in ANSI format (MODE 24576%).

When the search is successful, the file MARKIE is available for input, via GET statements. Remember, since the file is open for input only, attempting to execute PUT statements results in a PROTECTION VIOLATION error.

A subsequent CLOSE statement rewinds the tape (MODE 64%).

2.2 THE FILE STRUCTURED MAGTAPE OPEN FOR OUTPUT

The OPEN FOR OUTPUT statement searches for a specified file on a designated magtape unit. A BASIC-PLUS program executes the statement so that it can subsequently perform output to the file. For example:

```
10  OPEN "MT0:ABC" FOR OUTPUT AS FILE N%, MODE M%
```

The system associates magtape unit 0 with the internal channel designated by N% and searches for the file ABC on the current account according to the value M% in the MODE specification.

If the file is not found, the system writes a magtape label record for the file at the logical end of tape and leaves the unit open with write access only. A PUT statement subsequently executed on channel N% writes the channel's buffer to the magtape. Since the file is open solely for output, a GET statement executed on channel N% generates the PROTECTION VIOLATION error (ERR = 10).

The search is successful when the specified file is located. The value of M% in the MODE specification determines how the system searches for the file and acts upon the file when it is found. The following meanings can be attached to the MODE values used in an OPEN FOR OUTPUT statement. The MODE value used can be the sum of any combination of these single values, as long as they are not mutually exclusive.

MODE	Meaning
0%	Read record at current tape position.
2%	Do not rewind tape when system searches for the magtape file.
16%	Write over existing file. (Destroy any subsequent files currently on the tape.)
32%	Rewind tape before searching for the magtape file.
64%	Rewind tape upon executing the CLOSE statement.
128%	Open for append.
512%	Write new file label group without searching.
16384%	Search for a DOS formatted label.
24576%	Search for an ANSI formatted label.

2.2.1 Searching for a Record

Omitting the MODE specification or using a MODE 0% specification reads the tape at its current position. If the system finds a label record and its file name and account match those of the file specified in the OPEN statement, the system generates the NAME OR ACCOUNT NOW EXISTS error (ERR = 16). No match causes the system to rewind the tape and to check successive magtape label records until either a match is made or until the logical end of tape is detected. If the system detects the logical end of tape, the search is unsuccessful. As a result, the system backspaces over the logical end of tape, writes a label record for the file, and allows write access to the file. The system does not rewind the tape when the program executes a CLOSE statement on channel N%.

2.2.2 Writing A Record

As mentioned before, a search is successful when the system finds the specified file on the magtape. The NAME OR ACCOUNT NOW EXISTS error occurs when this happens. This is a precaution to prevent the user from writing a file at this point. (Doing so will write over the current file and erase all subsequent files on the tape.) A value of 16% included in the MODE specification suppresses this error message and causes the system to write over an existing file on magtape.

<u>NOTE</u>

MODE 16% causes any files following the overwritten file to be lost.

When 16% appears alone in the MODE specification, the system initially reads the magtape at its current position. If the system finds a label record and the file name in the label record matches the file name in the OPEN FOR OUTPUT statement, it backspaces over the label record, writes a new label record over the existing label and allows the user program write access to the file. If the logical end of tape is at the current position, the system backspaces one record and writes a new label record and allows write access to the file. No match causes the system to rewind the tape and to check label records until either the file is located or until the logical end of tape is detected. Detecting the logical end of tape before locating the file causes the system to backspace one record and write a tape label for the file and to allow the user write access to the file.

A CLOSE statement for a magtape open with MODE 16% writes trailing EOF records, backspaces two records, but does not rewind the tape.

When 512% is included in the value for the MODE option, the system writes a label record at the current tape position. No label record reading occurs. The system simply writes a new label record, erasing all subsequent files on the tape. Only the value 32%, which causes the tape to rewind (see Section 2.2.4), takes precedence over 512%. Therefore, when 512% is used in conjunction with any combination of values, not including 32%, the system writes a record label at the current tape position.

NOTE

> Any MODE value which includes 512% causes files following an overwritten file to be lost. The overwritten file is always the one at which the tape is currently positioned except when 32% is also included in the MODE value.

The FILESIZE and CLUSTERSIZE options have effect only when the file specified in the OPEN FOR OUTPUT statement is in ANSI format. The general form of this statement with options is:

```
          OPEN "MT0:ABC" FOR OUTPUT AS FILE N%, FILESIZE P%,
               CLUSTERSIZE Q%, MODE M%
```

The system associates magtape unit 0 with the internal channel designated by N%. The value P% in the FILESIZE option designates the block size and the value Q% in the CLUSTERSIZE option designates the record size, format and file characteristics. See Section 2.8.8 for explanations of format and file characteristics.

2.2.3 Extending A File

When 128% is included in the value for the MODE option, the system attempts to open an existing file and append information to it. The file must already exist; if it does not exist, a CAN'T FIND FILE OR ACCOUNT error (ERR = 5) occurs. The file must also be the last file on the tape before the logical end of tape. If it is not the last file on the tape, the system cannot locate the trailing EOF records and a PROTECTION VIOLATION error (ERR = 10) occurs. As is the case for all other MODE option values, 128% can be used alone or in conjunction with any combination of values.

2.2.4 Rewinding The Tape

As mentioned before, MODE 0% reads the tape at its current position. If the file name specified in the OPEN statement does not match the record label, the system automatically rewinds the tape to the first record label and begins reading labels file by file.

To override this automatic rewind feature, include the MODE value 2% in the OPEN statement. In this case, the system reads the tape from its current position and, if no match occurs, continues reading label records from that position forward until either the search is successful or until the logical end of tape is detected. The system does not rewind the tape when it performs a CLOSE operation. MODE 32% rewinds the tape to the first record label before reading any label. Once again, no match causes the system to check successive label records until the file is found or until the logical end of tape is detected. The system does not rewind the tape when it performs the CLOSE operation on channel N%.

Finally, including MODE value 64% rewinds the tape when a CLOSE statement is executed on channel N%.

2.2.5 DOS and ANSI Format Labels

Including MODE 16384% in the OPEN FOR OUTPUT statement searches for a magtape file whose file name is specified. In addition, the file must be written in DOS format (i.e., preceded by a DOS label) for the search to be considered successful.

When the user omits the MODE value 16384%, the system assumes record labels on the tape, either DOS or ANSI, are in the format specified by the system manager when he starts time-sharing.

Including MODE 24576% in the OPEN FOR OUTPUT statement searches for the magtape file whose file name is specified. In this case, the file must be written in ANSI format (i.e., preceded by an ANSI label) for the search to be considered successful. If the file is found, the system returns the NAME OR ACCOUNT NOW EXISTS error (ERR = 16).

The system reads the tape from its current position. If the file is not found, it rewinds the tape and reads label by label until it finds the correct file. If the logical end of tape is detected, the system automatically backspaces over two EOF records, writes an ANSI label record for the file, and allows write access to the file.

Once again, omitting the MODE value 24576% in the OPEN FOR OUTPUT statement assumes record labels are in the default format originally specified by the system manager.

2.2.6 Example Of OPEN FOR OUTPUT Statement

The values for MODE mentioned above can be combined in any combination as long as they are not mutually exclusive. Consider the following line:

```
10 OPEN "MT0:LLL317" FOR OUTPUT AS FILE 2%, MODE 16466%
```

This line opens the file LLL317 on magtape unit 0 and associates it with channel 2%. MODE 16466% is actually the sum of MODE 2% + 16% + 64% + 16384%.

When the above line is executed, the system determines whether the current record label is in DOS format (MODE 16384%). If the file is not found, the system does not rewind the tape (MODE 2%), but instead continues to search for labels in DOS format from the next record on. If the correct label record is found (i.e., LLL317 exists), the system backspaces one record and writes the new label over the existing label (MODE 16%). If the logical end of tape is found first, the system backspaces one EOF record and writes the new label, allowing write access to the new file.

Once the new record label is written, the file LLL317 is available for output, via PUT statements. Remember, since the file is open for output only, attempting to execute GET statements results in a PROTECTION VIOLATION error.

A subsequent CLOSE statement rewinds the tape (MODE 64%).

2.3 THE FILE STRUCTURED MAGTAPE OPEN

The OPEN statement performs an OPEN FOR INPUT operation for a designated file on a specific magtape unit. For example,

```
10  OPEN "MT0:ABC" AS FILE N%, MODE M%
```

The system associates magtape unit 0 with the internal channel designated by N% and searches for the file ABC on the current account as if an OPEN FOR INPUT statement were specified with M% in the MODE specification. An OPEN statement without a MODE specification is treated as if MODE 0% had been given. If the OPEN FOR INPUT operation succeeds, the program has read access to the file on the channel's buffer.

An unsuccessul OPEN FOR INPUT operation causes the system to perform an OPEN FOR OUTPUT operation using the MODE M% specification. The OPEN statement is not a recommended method for processing magtape since the program cannot immediately determine which type of OPEN statement was executed.

2.4 THE FILE STRUCTURED MAGTAPE CLOSE

The CLOSE statement terminates processing of a magtape file. If the file is open for input, the system skips to the end of the file (if it is not already there), and frees the buffer space for other usage within the program. If the file is open for output and the file label is in ANSI format, the system writes a trailer label group (see Appendix A). The system writes three EOF records to mark the logical end of tape, regardless of the file label format. It then backspaces the tape over two of the EOF records to position the tape for subsequent output, and frees the buffer space for other usage within the program.

Additionally, the system rewinds the tape if the value 64% was included in the MODE specification. Unless 64% is specified, the system does not rewind the tape.

2.5 THE NON-FILE STRUCTURED MAGTAPE OPEN

In non-file structured processing there are no special file label records written on the tape. Essentially, the system passes the data directly between the magtape and the user program. Tapes of any format can be read or written with non-file structured magtape operations, as long as the program is set up to handle the actual tape format correctly. Only records 14 bytes or longer can be accessed. Attempting to write a record shorter than 14 bytes results in the ILLEGAL BYTE COUNT FOR I/O error (ERR = 31).

In the OPEN statement, only the magtape unit is specified to indicate non-file structured processing; no filename is included. There are three types of OPEN statements, as before. These are:

```
100  OPEN "MT0:" FOR INPUT AS FILE 1%

               or

100  OPEN "MT0:" AS FILE 1%
```

OPEN FOR INPUT and the simple OPEN statements are equivalent. No magtape movement occurs and both reading and writing of records is permitted. The third form is slightly different:

```
100  OPEN "MT0:" FOR OUTPUT AS FILE 1%
```

In this example, the OPEN FOR OUTPUT permits writing only. This is the normal way of opening a magtape for writing.

2.6 THE NON-FILE STRUCTURED MAGTAPE CLOSE

CLOSE has no special action on non-file structured magtapes unless OPEN FOR OUTPUT was used. On a magtape that was OPEN FOR OUTPUT, the CLOSE statement causes trailer EOF records to be written, followed by backspacing over two of these EOF's, to position the tape correctly for subsequent output operations.

In any case, if the magtape was open for non-file structured processing, it is not rewound on CLOSE.

2.7 THE MODE SPECIFICATION IN NON-FILE STRUCTURED PROCESSING

The MODE specification with non-file structured magtape processing can be used with either 7-track or 9-track devices. When used with 7-track drive, MODE indicates both tape density and parity. When used with a 9-track TU10 drive, MODE indicates parity only, since a 9-track TU10 drive reads and writes only at 800 BPI. When used with a 9-track TU16 drive, MODE also can indicate 1600 BPI phase encoded mode.

MODE in the OPEN statement is evaluated by the following algorithm:

```
MODE E+D*4+P
```

where:

E (phase encoded) in bits per inch (BPI) is:

256 =	1600 BPI, phase encoded mode
0 =	values of D and P (see below)

D (density) in bits per inch (BPI) is:

0 =	200 BPI (7-track only)
1 =	556 BPI (7-track only)
2 =	800 BPI (7-track only)
3 =	800 BPI, dump mode

P (parity) is:

0 =	odd parity
1 =	even parity

If mode is not specified in the OPEN statement, the tape is processed in odd parity at 800 BPI dump mode, or with a MODE of 0%+3%*4%+0%.

Dump mode indicates that the system dumps from memory the actual representation of the data. Since there are 8 bits in one PDP-11 byte, 9-track drives, which have 8 data tracks, always operate in dump mode. For a 7-track drive, which contains space for only six data bits per frame, the dump mode means one PDP-11 8-bit byte is written to and read from two frames on the tape. The system uses one tape frame to hold bits 0 through 3 of a byte and a second frame to hold bits 4 through 7 of the byte.

Other values for density are in non-dump mode and apply only to 7-track drives. On output operations, the system writes bit 0 through 5 of each PDP-11 8-bit byte to a frame on the tape. Two bits, 6 and 7, are lost on each write operation. On input operations, the system transfers the data bits of a frame into bits 0 through 5 of a PDP-11 byte and sets bits 6 and 7 to zero. In this manner, bit formats other than the 8-bit byte format can be read from and written to 7-track magtape.

Effectively, MODE for a 9-track TU10 drive can only be a 0 or 1 since the system operates at 800 BPI dump mode with 9-track TU10 drives. If any other values are used, the system recognizes only the parity specification.

MODE for a 9-track TU16 drive can indicate 1600 BPI phase encoded operation. Parity is always odd for phase encoded operation.

To allow read and write access to a tape, use the OPEN or OPEN FOR INPUT statement. For example:

```
          OPEN "MT0:" AS FILE 1%, MODE 5%

                        or

          OPEN "MT0:" FOR INPUT AS FILE 1%, MODE 5%
```

Either statement makes the tape on the 7-track drive (set at unit 0) available for execution of GET and PUT statements on channel 1%. The system accesses tape with a density of 556 BPI and even parity. The system performs no tape positioning nor status checking. The user must perform such operations using the MAGTAPE function described in Section 2.8.

To allow only write access to a tape, use the OPEN FOR OUTPUT. For example:

```
          OPEN "MT1:" FOR OUTPUT AS FILE 1%, MODE 12%
```

If the unit is write locked (the write enable ring on the reel is missing), the system generates the DEVICE HUNG OR WRITE LOCKED error (ERR = 14) and does not open the device. Otherwise, the statement makes the tape on unit 1 available for execution of PUT statements on channel 1%. Since the device is open solely for write access, an attempt to execute a GET statement on the channel causes the PROTECTION VIOLATION error (ERR = 10). The system writes records in odd parity at a density of 800 BPI, dump mode. The user program must check the status of the device and control the device by use of the MAGTAPE function described in Section 2.8.

MAGTAPE

To read and write records larger than 512 bytes on magtape, include the RECORDSIZE option in the OPEN statement as described in the BASIC-PLUS Language Manual in Section 9.2.1. Subsequent GET operations read records of maximum size equal to RECORDSIZE. RECOUNT contains the actual number of bytes read. PUT operations which are not modified by the COUNT modifier write records whose size is specified in the RECORDSIZE option.

To write records smaller than 512 bytes, use the COUNT option described in the BASIC-PLUS Language Manual in Section 12.3.2. If smaller records are read, the RECOUNT variable contains the number of bytes.

2.8 THE MAGTAPE FUNCTION IN NON-FILE STRUCTURED PROGRAMMING

The MAGTAPE function provides flexibility in non-file structured processing by permitting the program control over all magtape functions. The general form of the MAGTAPE function is as follows:

```
          I% = MAGTAPE (F%,P%,U%)
```

where:

F%	is the function code (1 to 9)
P%	is the integer parameter
U%	is the internal channel number on which the selected magtape is open
I%	is the value returned by the function

The effect of the MAGTAPE function is determined by the function code, F%. These functions are described in the sections that follow. In all examples in these sections, assume that magtape unit 1 has been opened on internal channel 2. That is, prior to executing the MAGTAPE function, the following statement was executed.

```
          100  OPEN "MT1:" AS FILE 2%
```

The following discussion of each of these functions includes the word IMMEDIATE or WAIT. IMMEDIATE indicates that the monitor initiates the action and returns control to the program immediately; WAIT indicates that the program must wait for the operation to be completed before continuing.

2.8.1 Off-Line (Rewind and Off-Line) Function

IMMEDIATE

Function code	= 1
Parameter	= unused
Value returned	= 0

The Off-Line function causes the specified magtape to be rewound and set to OFF-LINE (thus clearing READY). For example:

```
          200  I% = MAGTAPE(1%,0%,2%)
```

rewinds and sets the magtape open on internal channel 2 to OFF-LINE.

2.8.2 WRITE End-of-File Function

WAIT

Function code	= 2
Parameter	= unused
Value returned	= 0

The WRITE End-of-File function writes one EOF record at the current position of the magtape. For example:

```
          200  I% = MAGTAPE(2%,0%,2%)
```

writes an EOF on the magtape that is open on internal channel 2.

2.8.3 Rewind Function

IMMEDIATE

```
     Function code   = 3
     Parameter       = unused
     Value returned  = 0
```

The Rewind function rewinds the selected magtape. For example:

```
          200  I% = MAGTAPE(3%,0%,2%)
```

rewinds the magtape open on internal channel 2. (This function does not cause the magtape to be set to OFF-LINE.)

2.8.4 Skip Record Function

WAIT

```
     Function code   = 4
     Parameter       = number of records to skip (1 to 32767)
     Value returned  = number of records not skipped
                       (0 unless EOF or physical EOT is encountered)
```

The Skip Record function advances the magtape down the tape. The tape continues to advance until either the desired number of records is skipped (in which case the value returned by the function is 0) or an EOF record is encountered (in which case the value returned is the specified number of records to skip minus the number actually skipped).(1) For example, to skip from the current tape position to just past the next EOF, use the following function:

```
          200 I% = MAGTAPE(4%,32767%,2%)
```

This assumes there are fewer than 32767 records before the next EOF. In Section 2.8.7, a more complex example using the MAGTAPE function shows how to skip an entire file regardless of the number of records.

2.8.5 Backspace Function

WAIT

Function code	= 5
Parameter	= number of records to backspace (1 to 32767)
Value returned	= number of records not backspaced (0 unless EOF or beginning-of-tape is encountered)

The Backspace function is similar to the Skip function, except that tape motion is in the opposite direction. The beginning-of-tape (BOT or Load Point) as well as EOF records can cause premature termination of the Backspace operation (in which case the value returned is the specified number of records to backspace minus the number actually backspaced).(2) The BOT is neither skipped nor counted as a skipped record. For example:

```
200 I% = MAGTAPE(5%,1%,2%)
```

backspaces one record on the magtape opened on internal channel 2, unless the tape was already at BOT.

2.8.6 Set Density And Parity Function

IMMEDIATE
(Insignificant)

Function code	= 6
Parameter	= E+D*4+P
Value returned	= 0

where:

E = Phase Encoded

256 = 1600 BPI, phase encoded mode
0 = values of D and P (see below)

D = Density

0 = 200 BPI (7-track only)
1 = 556 BPI (7-track only)
2 = 800 BPI (7-track only)
3 = 800 BPI, dump mode

P = Parity

0 = odd
1 = even

----

(1)The system counts the EOF record as a record skipped.

(2)The system counts the EOF record as a record actually backspaced.

Magtape drives are set to the system default density/parity when first assigned (with an ASSIGN statement) or when first opened. This function changes the density and/or parity of a magtape drive according to the value given as the parameter. The function interprets the parameter exactly as MODE value (see Section 2.7). For example,

```
10 OPEN "MT0:" AS FILE 2%

20 I%=MAGTAPE(6%, 2%*4%+1%, 2%)
```

changes the density and parity of the 7-track magtape drive open on channel 2 to 800 BPI, even parity, non-dump mode. The density and parity specified in the parameter is in effect until channel 2 is closed. The system sets 1% to 0 to indicate successful completion.

By adding 8192% to the parameter value (making it 8192%+2%*4%+1%, in the above example), the new density/parity setting is in effect even after the associated channel has been closed. A subsequent OPEN statement, associating any channel number with magtape unit 0, opens it with that new density/parity setting automatically. Only a DEASSIGN statement to a previously assigned unit or another parameter specified in this function can revert the density/parity setting for the magtape unit back to the system default value.

The sample routine shown below in immediate mode sets magtape unit 2 to 800 BPI, dump mode, odd parity, using DOS labels. Once channel 3 is closed, in this example, the new density/parity setting is now in effect and remains in effect until a DEASSIGN statement is executed on magtape unit 2.

```
ASSIGN MT2:.DOS

OPEN "MT2:" AS FILE 3%

I%=MAGTAPE(6%, 8192%+3%*4%,3%)

CLOSE 3%
```

2.8.7 Tape Status Function

IMMEDIATE
(Insignificant)

Function code	= 7
Parameter	= unused
Value returned	= status

The Tape Status function returns the status of the specified magtape as a 16-bit integer, with certain bits set, depending on the current status. The format is shown in Table 2-1. The example shown below obtains the status of the magtape opened on internal channel number 2:

```
200  I% = MAGTAPE(7%,0%,2%)
```

Table 2-1
Magtape Status Word

Bit	Test	Meaning
15	I% < 0%	Last command caused an error
14-13	(I% AND 24576%)/8192%	Density: 0 = 200 BPI 1 = 556 BPI 2 = 800 BPI 3 = 800 BPI, dump mode
12	(I% AND 4096%) = 0% (I% AND 4096%) <> 0%	9-track tape 7-track tape
11	(I% AND 2048%) = 0% (I% AND 2048%) <> 0%	Odd parity Even parity
10	(I% AND 1024%) <> 0%	Magtape is physically write locked
9	(I% AND 512%) <> 0%	Tape is beyond end-of-tape marker
8	(I% AND 256%) <> 0%	Tape is a beginning-of-tape (Load Point)
7	(I% AND 128%) <> 0%	Last command detected an EOF
6	(I% AND 64%) <> 0%	The last command was READ and the record read was longer than the I/O buffer size (i.e., part of the record was lost).
5	(I% AND 32%) <> 0%	Unit is non-selectable (OFF-LINE)
4	(I% AND 16% <> 0%	Unit is TU16
3	(I% AND 8%) <> 0%	1600 BPI, phase encoded mode (for TU16)
2-0	(I% AND 7%)	Indicates last command issued: 0 = OFF-LINE 1 = READ 2 = WRITE 3 = WRITE EOF 4 = REWIND 5 = SKIP RECORD 6 = BACKSPACE RECORD

When the value of I% returned is 17,409 decimal (or 42001 octal), the magtape is 800 BPI, 9-track, odd parity, write protected, and the last command issued was READ. This can be determined by testing the value of I%, bit by bit, against Table 12-1, as follows:

```
     I% = 17,409 (decimal)

        =  4   2   0   0   1   (octal)

        = 100 010 000 000 001  (binary)
```

The test for density uses bits 14 and 13: (I% AND 24576%)/8192%

```
          I%  100 010 000 000 001

AND 17409%  110 000 000 000 000

     Result  100 000 000 000 000
```

Dividing the result of (I% AND 24576%) (in this case, that result is 16384%) by 8192%, the quotient can equal 0, 1, 2, or 3. In this case, 16384/8192 = 2, indicating that the tape density is 800 BPI.

The results of (I% AND 4096%) and (I% AND 2048%) are both zero, indicating a 9-track tape with odd parity. (I% AND 1024%) results in a non-zero number in this case, so the magtape is physically write locked.

Bits 9 through 3 of I% in this example are all zero, but (I% AND 7%) results in 1%, indicating bit 0 is set and the last command issued was READ.

Another magtape status function can advance to the next EOF (i.e., skip over the current file). The Skip Record function can do this unless the file is longer than 32,767 records -- in which case several skip record functions must be executed -- or an EOT is detected within a file. The following statements execute a Skip Record function until the next EOF is encountered.

```
200  I% = MAGTAPE(4%,32767%,2%):
       IF (MAGTAPE(7%,0%,2%) AND 128%) = 0% THEN 200
```

2.8.8 Return File Characteristics Function

IMMEDIATE
(Insignificant)

Function code	= 8
Parameter	= unused
Value returned	= file characteristics

This function returns the status of the specified file structured magtape as a 16-bit integer, with certain bits set depending on the current file characteristics. The format is shown in Table 2-2. Non-zero integers are returned for DOS files; zero is returned for ANSI files.

For example, to obtain the characteristics of a file on a magtape opened on channel 2:

```
400  I% = MAGTAPE(8%,0%,2%)
```

When the value of I% returned is 16,464 decimal (40120 octal), the magtape file is in ANSI format F, carriage control is embedded 'M', and the record length is 80 bytes. This can be determined by testing the value of I%, bit by bit, against Table 2-2, as follows:

```
I% = 16,464 (decimal)

   = 0  4   0   1   2   0   (octal)

   = 0 100 000 001 010 000 (binary)
```

The test for ANSI format type is (SWAP%(I%) AND 192%)/64%, where 192% = 128% + 64%.

```
SWAP%(I%) 0 101 000 001 000 000

AND 192%             11 000 000

  Result              1 000 000
```

Dividing the result of SWAP%(I%) AND 192% (in this case, that result is 64%) by 64%, the quotient equals 64%/64% = 1, indicating that the magtape is under ANSI format F.

The result of (I% AND 12288%)/4096% is 0 in this example, indicating that the carriage control is embedded 'M'.

Finally, the result of (I% AND 4095%) yields 80 in this case, so the record length is 80 bytes.

Table 2-2
Magtape File Characteristics Word

Bit	Test	Meaning
15-14	(SWAP%(I%)AND 192%)/64%	ANSI format: 0 = U (undefined) 1 = F (fixed length) 2 = D (variable length) 3 = S (spanned(1))
13-12	(I% AND 12288%)/4096%	Format U operation: 0 (default) Format D, S and F operation: 0 (carriage control embedded 'M') 1 (FORTRAN carriage control 'A') 2 (implied LF/CR before record' ')
11-0	I% AND 4095%	Format U operation: 0 = (default) Format F operation: Record length Format D operation: Maximum record length Format S operation: unused

2.8.9 Rewind On CLOSE Function

IMMEDIATE
(Insignificant)

```
Function code    = 9
Parameter        = unused
Value returned   = 0
```

The Rewind on CLOSE function causes the selected magtape to be rewound when the CLOSE statement is executed. For example:

```
I% = MAGTAPE(9%,0%,2%)
```

rewinds the magtape open on internal channel 2 when CLOSE is executed from a program or when CLOSE is executed in immediate mode.

The Rewind on CLOSE function must be used after the OPEN statement and before the CLOSE statement. This function overrides all MODE specifications which, in the OPEN statement, instruct the system not to rewind on closing the file. Once the Rewind on CLOSE function is executed, it cannot be cancelled.

----

(1)ANSI format S is not supported by RSTS/E systems.

2.9 MAGTAPE ERROR HANDLING

It is important to consider details of the system's handling of magtape error conditions. These are: parity error, record length error, off-line (not ready) error and write lock error.

2.9.1 PARITY (Bad Tape) ERROR

If an error is detected on a read attempt, the system attempts to re-read the record up to 15 times. If the error condition persists, a USER DATA ERROR ON DEVICE error occurs. In this case, the read has been completed, but the data in the I/O buffer cannot be considered correct. On an output operation, if the first attempt to write a record fails, the system attempts to rewrite the record up to 15 times using write with Extended Interrecord Gap to space past a possible bad spot on the tape. If the error condition persists, a USER DATA ERROR ON DEVICE error occurs. In both cases, the tape is positioned just past the record on which the error occurred.

2.9.2 RECORD LENGTH ERROR

This error can occur only during a read operation when the record on the magtape is longer than the I/O buffer size, as determined by the OPEN statement. The extra bytes in the record are not read into memory but are checked for possible parity errors. If a parity error occurs, an error message is returned to the user program, and bit 6 of the magtape status word is set. Therefore, if a program is reading records of unknown length from magtape, it is necessary to check for possible record length errors after every read operation. This can be done as follows:

```
          200  PRINT "RECORD TOO LONG" IF MAGTAPE (7%,0%,2%) AND 64%
```

Note that in the above example if bit 6 is set in the magtape status word the IF condition tests as TRUE. The error, MAGTAPE RECORD LENGTH ERROR (ERR = 40), occurs when the tape block is too long, in file-structured or non-file structured magtape.

2.9.3 OFF-LINE ERROR

The status of the magtape unit is determined by testing bit 5 of the returned value of the magtape status function shown in Table 2-2. If bit 5 is set, the magtape unit is off-line. A MAGTAPE SELECT ERROR (ERR = 39) occurs if an attempt to access an off-line drive is made.

2.9.4 WRITE LOCK ERROR

Attempting any write operation on a magtape that is physically write locked (i.e., a tape that does not have the write enable ring inserted) results in a "DEVICE HUNG OR WRITE LOCKED" error.

2.10 THE KILL AND NAME AS STATEMENTS

The KILL and NAME AS statements described in the BASIC-PLUS Language Manual are applicable only to disk and DECtape files; they cannot be used with magtape files.

CHAPTER 3

LINE PRINTER

3.1 SPECIAL CHARACTER HANDLING

Certain characters have special significance on line printer output. Table 3-1 summarizes LP11 and LS11 operation under RSTS/E for each of these special characters.

Table 3-1
LS11 and LP11 Commands

Character	LS11 Function	LP11 Function
CHR$(7)	BELL (A 2-second audible tone)	Ignored by driver
CHR$(9)	TAB - Horizontal Tab 1. Spaces over to next tab position (columns 1, 9, 17, 25, etc.)	TAB - Horizontal Tab 1. Spaces over to next tab position (columns 1, 9, 17, 25, etc.)
CHR$(10)	LF - Line Feed 1. Print line 2. Carriage return 3. Advances paper one line	PF - Paper Feed 1. Print line 2. Carriage return 3. Advances paper one line
CHR$(11)	VT - Vertical Tab 1. Advances paper to the next hole position in Channel 5	VT - Vertical Tab 1. Ignored by driver

Table 3-1 (Cont.)
LS11 and LP11 Commands

Character	LS11 Function	LP11 Function
CHR$(12)	FF - Form Feed 1. Print line 2. Carriage return 3. Advances paper to the next hole position in Channel 7 (see Section 3.2)	FF - Form Feed 1. Print line 2. Carriage return 3. Advances paper to the third line of the next form (hardware top of form, see Section 3.2)
CHR$(13)	CR - Carriage Return 1. Print line 2. Carriage return 3. No line feed (may be used for overprint)	CR - Carriage Return 1. Print line 2. Carriage return 3. No line feed (may be used for overprint)
CHR$(14)	ELONG - Elongated Character 1. Doubles the horizontal printing axis	ELONG - Elongated Character 1. Ignored by driver
CHR$(17)	SEL - Select 1. Allows the software to put the printer on-line	SEL - Select 1. Ignored by driver
CHR$(19)	DSEL - Deselect 1. Allows the software to put the printer off-line	DSEL - Deselect 1. Ignored by driver
CHR$(96) to CHR$(126)	1. Lower case printing characters, converted to upper case	1. Lower case printing characters, converted to upper case except on an upper case/lower case printer.
CHR$(127)	DEL - Delete 1. Ignored by driver	DEL - Delete 1. Ignored by driver

3.2 SPECIAL OPERATIONS

The MODE specification in the OPEN statement allows the user to control line printer operations. For example:

```
OPEN "LP:" AS FILE F%, MODE M%
```

The system associates line printer unit 0 with the channel designated by F%. The value of M% in the MODE specification determines the actions the system performs at the line printer. The following MODE values generate the action described at the line printer unit.

MODE Value	Action
1 to 127	Sets form length in number of lines per page for software formatting (512%) and/or automatic page skip (2048%). This is the LPFORM option.
128	Change the character 0 to the character O.
256	Truncate lines which are longer than the unit was configured for. This action is done in place of printing the remainder of the line on the next physical line on the page.
512	Enable software formatting. Forms control characters are $\geq$128.
1024	Translate lower case characters to upper case characters. Applies only to upper and lower case line printers.
2048	Skip six lines (i.e., over perforation line) at the bottom of each form.
4096	Moves paper to top of hardware form. CHR$(12%). (See discussion, below.)

The MODE value 512% allows a program to control non-standard length forms in the line printer(1). To accomplish this action, the program must include a MODE value between 1 and 127 to indicate the number of lines per page on the printer. For example,

```
          100  OPEN "LP0:" AS FILE 1%, MODE 542%
```

The statement sets the form length to 30 lines per page (512%+30, or 542%). If neither the 512% nor the value for form length is given, the system uses 66 lines per page as the form length. Lines are numbered from 1 to the length specified. Thus, in this example, lines are numbered from 1 to 30.

As a result of enabling the software formatting with MODE 512%, certain special characters that the program sends to the line printer determine the number of the line on which the system prints data. The system skips to this line by sending the proper number of LINE FEED characters to the printer. It determines the line on which to print by subtracting 127 from the decimal value of the special character the program sends to the printer. For example,

```
          PRINT #1, CHR$(147%);
```

This statement causes the system to evaluate the difference between 147 and 127. If the difference is greater than the page length

----

(1)The hardware option on the LP11 High Speed Printer to automatically skip over perforations must be disabled for this option to execute properly.

specified in the MODE value or more than 66 when no page length is specified, the system ignores it. If the difference is less than the page length in effect but greater than the current line number, the printer skips to that line number on the current page. If the difference is less than or equal to the number of the current line, the system skips the printer to the appropriate line on the next page.

NOTE

> To enable the program to properly perform software formatting of print lines using special characters, the user must load the paper in the line printer with the top of form at the arrows and with the tractors set at their top of form position. See the RSTS-11 System User's Guide for line printer operating procedures.

The system treats characters whose values lie between 0 and 127 as the standard ASCII equivalents as shown in Appendix D. In addition, a LF character sent to the printer advances the form to the first print position of the next line (i.e., LF implies CR). If a form length other than 66 is specified but MODE 4096% is not specified, a CHR$(12) in the data passed by the program to the line printer translates to a sufficient number of line feed characters to move the page to the top of (software) form. If the form length is 66 or zero (the default value which results in 66) or MODE 4096% is specified, CHR$(12) remains untranslated and positions the printer paper at the top of the hardware form.

A value of 128 in the MODE specification causes the system to print all 0 characters as O characters. This feature is valuable in commercial applications where there can be no possibility for confusion. For example,

```
10 OPEN "LP0:" AS FILE 1%, MODE 706%
```

The statement indicates software formatting (512%) and translation of 0 to O (128%) is to be performed on line printer unit 0 with a form length of 66. That is, MODE 706% is equivalent to MODE 512%+128%+66%.

To truncate lines greater than the width of the line printer, the user program includes 256% in the MODE value. For example,

```
10 OPEN "LP0:" AS FILE 1%, MODE 962%
```

The statement implements the MODE value of 66%, 128%, 256% and 512%, on line printer unit 0 and discards excess characters from each line printed (MODE 256%). Without 256% in MODE, the system prints excess characters on a second physical line.

To translate lower case characters to upper case characters, the user program includes 1024 in MODE. For example,

```
10 OPEN "LP0:" AS FILE 1%, MODE 1986%
```

This statement implements the MODE values 66%, 128%, 256% and 512% and, in addition, causes the system to translate all characters with representations between CHR$(96) and CHR$(122) to their equivalents between CHR$(6%) and CHR$(90). This feature is always set for an upper case only printer.

To skip six lines at the bottom of each form, the user program includes 2048 in MODE. For example,

```
10 OPEN "LPO:" AS FILE 1%, MODE 4034%
```

The statement implements the MODE values 66%, 128%, 256%, 512%, 1024%, and also skips six lines when the system advances the page to top of form. With this value in effect, the system does not print on the last six lines of each form. This feature is useful when generating continuous listings to be placed in horizontal binders. If the user loads the line printer as such that the top of form is the third physical line on the page, the system leaves three blank lines at the beginning of the next page. When the listings are subsequently placed in binders, printed material is located three lines from the perforations of the page to facilitate easy access.

The system handles the actual transfer of data from user storage to the line printer by means of an intermediate storage called small buffers. This allows the faster computational process to continue unhindered by the slower output action at the line printer. Thus, for each output request in a user program, the system transfers the related program data from program storage to small buffers. At the same time, at its own speed, the line printer software extracts the data from the intermediate small buffers and performs the output to the device.

An error condition at the line printer causes the system to interrupt the transfer of data both from the buffers to the device and from the program to the buffers. Since any number of indeterminate events such as a ribbon jam or a paper tear can cause an error condition, the system retains the unprinted data in the buffers until either the error is cleared (the unit becomes ready again) and the user program executes a CLOSE operation.

The system checks the status of the line printer every ten seconds, and, upon detecting the ready condition again, continues output from the small buffers without loss of data. If the user program closes the line printer while the error is still pending, the system returns the small buffers to the pool without printing their contents. The data transferred from the program but not yet printed is lost.

If the user program disregards the error condition and continues processing, the system does not transfer more data to additional small buffers. No output occurs at the line printer while the error condition remains in effect.

To prevent loss of data, a user program must properly detect a line printer error condition and execute efficient error handling. The system indicates the line printer error by generating the DEVICE HUNG OR WRITE LOCKED error (ERR = 14). The first time this error is returned after an output request (e.g., PUT), the data is fully buffered by the monitor. No data is lost, but the buffered data cannot be sent to the printer due to the error condition. Every 10 seconds the monitor checks the printer's status. It will resume printing when the error condition is rectified. To prevent filling up monitor free space, subsequent output requests are returned immediately without any further data buffering and with ERR=14 while the error condition persists. When the output request returns without error, the printer error condition is cleared.

The sample program shown below demonstrates code which performs the following actions:

a. opens the line printer, inputs a line from the disk file, and performs output to the line printer, and

b. performs efficient error handling as described above.

```
10      !       HOUSEKEEPING
20      OPEN "DATA.DAT" FOR INPUT AS FILE 1
30      OPEN "LP0:" AS FILE 2, RECORD SIZE BUFSIZ(1%)
40      FIELD #1, BUFSIZ(1%) AS I$
50      FIELD #2, BUFSIZ(2%) AS O$
60      E% = 0%
70      ON ERROR GOTO 200
100     !       DATA MOVING LOOP
110     GET #1
120     C% = RECOUNT
130     LSET O$ = I$
140     PUT #2, COUNT C%
150     E% = O%
160     GOTO 100
200     !       ERROR HANDLING
210     IF ERR=11 AND ERL=110 THEN RESUME 400
220     IF ERR=14 AND ERL=140 THEN RESUME 300
230     ON ERROR GOTO 0
300     !       PRINTER ERRORS
310     IF E% THEN 350
320     !       FIRST TIME ERRORS (DATA WAS BUFFERED)
330     E% = -1%
340     GOTO 100
350     !       SUBSEQUENT ERRORS (DATA NOT BUFFERED)
360     PRINT "?LINE PRINTER ERROR?"
370     SLEEP 10%
380     GOTO 140
400     !        DONE
410     CLOSE 1, 2
999     END
```

CHAPTER 4

TERMINALS

Several features are available to facilitate processing on keyboard devices using Record I/O statements.

4.1 CONDITIONAL INPUT FROM A TERMINAL

Sometimes a program must execute an input request from a terminal without waiting for data to be available. (The terminal may be opened on a specific I/O channel or may be one of many terminals opened on one I/O channel - see Section 4.3.) Normally, the system stalls the program executing an input request until data is available in the keyboard input buffer (i.e, a line terminator is typed at the keyboard). To avoid stalling, the program can execute a statement similar to the following:

```
GET #1%, RECORD 8192%
```

If data is available from the terminal open on channel 1, the system makes it accessible to the program in the channel 1 buffer. The number of bytes read from the terminal input buffer is given by the RECOUNT variable. If no data is available, the system generates the USER DATA ERROR ON DEVICE error (ERR = 13). In both cases, the system reports the results immediately.

4.2 BINARY DATA OUTPUT AND INPUT

To perform binary data output to a terminal, either opened on its own I/O channel or opened as one of many terminals on one I/O channel, execute a statement of the following form:

```
PUT #N%, RECORD 4096%, COUNT M%

                or

PUT #N%, RECORD 1%, COUNT M%
```

The statement transfers the number of bytes specified by M% to the output buffer of the terminal open on channel N%. No special form of the output statement is required. As a result of specifying RECORD 4096% or RECORD 1% in the PUT statement, all output formatting on the terminal is disabled.

A user program can obtain binary input from a keyboard by executing the OPEN statement with a MODE 1% option. For example,

```
10  OPEN "KB6:" AS FILE N%, MODE 1%
```

The statement associates channel N% with keyboard number 6 in binary input mode. As a result, characters received are not echoed by the system and are not altered in any way. In this manner, a program can read binary data from a terminal paper tape reader, from the terminal itself, or from any device connected to the system through a keyboard interface.

To initiate a transfer of data, use the GET statement. For example,

```
GET #N%
```

The system transfers some number of characters from the keyboard open on channel N% to the buffer for that channel. If no data is available, the system stalls the operation until data is received from the keyboard. When data is received, the program is made runnable and the data is available in the buffer. The BASIC-PLUS program must execute GET statements frequently enough to avoid losing data from the transmitting device.

The number of characters received is always at least one and never more than the channel buffer size. The default buffer size for keyboards is 128 characters. The user can override the default buffer size by the RECORDSIZE option in the OPEN statement. The RECOUNT variable contains the actual number of characters received.

Since the system accepts and does not alter any characters received from a terminal open for binary input, typing ^C on such a terminal has no effect. For this reason, the system disables binary input mode under the following conditions.

a. The period for a WAIT statement expires

b. Executing a statement on channel zero when the user's keyboard is open for binary input

c. Executing an OPEN statement in normal mode on the device but on a different channel

d. Executing a CLOSE statement on any channel associated with a keyboard open for binary input

For condition a, the system disables binary input mode if time for a WAIT is exhausted. For example,

```
10  WAIT 10%
20  GET #1%
```

If the system does not detect data within 10 seconds on channel 1, which is open for binary input, it disables binary mode in addition to generating the KEYBOARD WAIT EXHAUSTED error (ERR = 15). The keyboard remains open for normal ASCII data transfers.

For condition b, the system disables binary input mode when the program executes a statement on channel 0 and the user's keyboard is open for binary input on a non-zero channel. For example,

```
10  OPEN "KB:" AS FILE 1%, MODE 1%
20  GET #1%
.
.
.
40  PRINT "MESSAGE";
```

The statement at line 10 opens the user's keyboard for binary input on a non-zero channel (channel 1). The statement at line 20 performs binary input from the keyboard. At line 40, however, the system executes a PRINT statement on the user's keyboard (channel 0) which disables binary input mode. The user's terminal remains open on channel 1 for normal ASCII data transfers.

For condition c, the system disables binary input on a channel if the program executes a normal OPEN on the same device but on a different channel. For example,

```
10  OPEN "KB6:" AS FILE 1%, MODE 1%
.
.
.
100  OPEN "KB6:" AS FILE 2%
```

As a result of statement 100, the system disables binary input on keyboard 6. If statement 100 contained MODE 1%, the system would open keyboard 6 for binary input on channel 2. Keyboard 6 would therefore be open for binary input on both channels.

For condition d, the system disables binary input if the program executes a CLOSE statement on any channel associated with a keyboard open for binary input. For example,

```
10 OPEN "KB6:" AS FILE 1%, MODE 1%
20 OPEN "KB6:" AS FILE 2%, MODE 1%
.
.
.
100 CLOSE 2%
```

At line 100, the CLOSE statement disassociates channel 2 from keyboard 6 but also disables binary input on channel 1. Keyboard 6 remains open in normal mode on channel 1.

The recommended method of using binary input mode is to open a device other than the user's terminal for binary input on any non-zero channel. The program interacts normally with the user's terminal by executing standard INPUT and PRINT statements and gathers data from the binary device on the non-zero channel by executing GET statements.

Since binary input disables all special character handling, the system cannot detect an end of file on a terminal transmitting binary data.

4.3 MULTIPLE TERMINAL SERVICE ON ONE I/O CHANNEL

The multiple terminal feature allows one program to interact with several terminals rather than merely having each terminal open for input or output. This feature is useful in applications such as order entry, inventory control, and query-response where the same function is performed on several terminals but a separate job for each terminal is undesirable or inefficient.

To implement control of several terminals, the BASIC-PLUS program must first establish a master terminal by opening a keyboard on a non-zero channel. Two forms of the OPEN statement are possible. For example,

```
          10  OPEN "KB:" AS FILE N%

                    or

          10  OPEN "KB4:" AS FILE N%
```

The first form associates channel N% with the current keyboard and defines it as the master terminal. The second form associates channel N% with keyboard number 4 and defines it as the master terminal.

The program exercises control of additional, or slave, terminals, through special forms of the Record I/O GET and PUT statements. The terminals must be reserved to the job but must not be open in the program. The user can establish the terminals as slave terminals with the ASSIGN immediate mode command before he runs the program. Alternatively, the program can assign these terminals by executing the number 10 SYS system function call to FIP. The program can control any number of these additional terminals up to the maximum number of terminals on the system.

To perform input and output operations, the program uses GET and PUT statements in a special manner. The RECORD option specifies a particular action and keyboard number.

A PUT statement of the following form performs output to a keyboard, either master or slave.

```
          10  PUT #1%, RECORD 32767%+1%+K%, COUNT N%
```

The variable K% in the RECORD option is the unit number of the keyboard to which output is directed. As a result, the number of characters specified by N% in the COUNT option is transferred from the buffer on channel 1 to the designated keyboard. The only special error which can occur is NOT A VALID DEVICE (ERR = 6), indicating that the terminal addressed is neither the master keyboard nor a slave keyboard reserved to the program. Other possible errors such as I/O CHANNEL NOT OPEN (ERR = 9) work in the standard fashion.

A GET statement of the following form performs input from a keyboard, either master or slave.

```
          10  GET #1%, RECORD 32767%+1%+K%
```

The variable K% in the RECORD option is the unit designator (keyboard number) of the terminal from which input is requested. The GET statement transfers the data from the terminal input buffer to the buffer for the designated channel. The first character in the buffer contains the number of the keyboard from which the input came. The total number of characters transferred, including the keyboard number, is given by the RECOUNT variable. The program accesses the data by use of the standard FIELD statement. Since the first character of the I/O buffer is the keyboard number, the length of the data input is given by RECOUNT-1%. If no input is available from the designated terminal, the USER DATA ERROR ON DEVICE error (ERR = 13) results. Since this error is recoverable, the program can execute an appropriate ON ERROR GOTO routine.

The following form of the GET statement requests input from any one of the multiple terminals.

```
          10  GET #1%, RECORD 32767%+1%+16384%+S%
```

If input is pending from any terminal, the contents of that terminal's buffer are transferred to the buffer for the designated channel. The first character in the buffer is the keyboard number of the terminal from which input came. As described above for input from a specific keyboard, the FIELD statement can access the sending keyboard number and the data sent. The variable S% can have the following values:

Value	Meaning
S%=0%	GET statement waits until input is available from any one of the terminals. The system waits indefinitely if no input is pending. When input is available, the system transfers the data and the program accesses the data as described above. A USER DATA ERROR ON DEVICE error (ERR =13) may occur due to a race condition with CTRL/C. No data is lost; simply re-issuing the GET statement continues operation.
1%<S%<255%	GET statement waits up to S% seconds for input from any terminal. If no input is available from any terminal in S% seconds, the USER DATA ERROR ON DEVICE error (ERR = 13) occurs.
S%=4096%	Binary data (see Section 4.2)
S%=8192%	If no input is pending from any of the terminals, the USER DATA ERROR ON DEVICE error (ERR = 13) occurs immediately (see Section 4.1).

A CTRL/C combination typed at any one of the slave terminals passes a CHR$(3) character to the program but does not terminate the program. The RECOUNT variable contains the value 2% representing the keyboard number and the CTRL/C character. The program can process the CTRL/C character as a special character. If CTRL/C is typed at the master terminal, the system terminates the program in the standard fashion.

A CTRL/Z combination typed at a master or at a slave terminal causes the END OF FILE ON DEVICE error (ERR = 11) to occur. The unit number of the keyboard causing the error is returned as the first character in the buffer on the channel. The value of the RECOUNT variable is meaningless.

When the value 4096% is also included in the RECORD option, binary data can be output using this multiple terminal service. For example,

```
100 PUT #N%, RECORD 32767%+1%+4096%+K%, COUNT M%
```

is used to output the number of bytes of binary data specified by M% to the keyboard whose unit number is the variable K%.

4.4 ESCAPE SEQUENCES

4.4.1 Output Escape Sequences

When sending an escape sequence to a terminal, use the value CHR$(155) (233 octal) for the escape character rather than CHR$(27) (033 octal). The system translates CHR$(27) to CHR$(36) (044 octal) which is the dollar sign ($) character. The system translates neither the CHR$(155) character nor the character following it. This process allows the terminal to interpret the escape sequence transmitted.

4.4.2 Input Escape Sequences

There are two I/O operating modes which are set by the TTYSET system program: NO ESC SEQ mode and ESC SEQ mode. In NO ESC SEQ mode, an incoming ESC character -- CHR$(27) (033 octal) -- acts as a delimiter. The system echoes a CHR$(36) (044 octal), which is the dollar sign ($) character.

In ESC SEQ mode, however, an incoming CHR$(27) merely sets a flag indicating that the next input character is special; no character is echoed when CHR$(27) is input in ESC SEQ mode. The next character input after ESC acts as a delimiter, but does not echo. This second character clears the previously set flag. The system then reverses the order of the last two characters and translates the CHR$(27) to CHR$(155) (233 octal).

For example, in ESC SEQ mode, if the two incoming characters are:

```
ESC P (CHR$(27),CHR$(80))
```

the system transposes this to:

```
P ESC (CHR$(80),CHR$(155))
```

In this way, the character input immediately after ESC is retained and can be used or tested by the user program. Some terminals (see following sections) and other devices respond to escape sequences.

4.5 VT05 CURSOR CONTROL AND SPECIAL CHARACTERS

The VT05 alphanumeric display terminal recognizes certain control characters that are not used on other terminals. These are summarized in Table 4-1.

Table 4-1
VT05 Special Display Characters

Character String	Meaning
CHR$(8)	BACKSPACE (move cursor left one character).
CHR$(11)	CURSOR DOWN (one line, same position).
CHR$(14)	Direct cursor control The next two characters give x- and y-coordinates of the new cursor position on the 20-line by 72-character VT05 screen. The characters following CHR$(14) are, first, CHR$(32+Y) and second, CHR$(32+X). Y is the y-coordinate (0 to 19), and X is the x-coordinate (0 to 71).
CHR$(24)	CURSOR RIGHT (one character)
CHR$(26)	CURSOR UP (one line, same position).
CHR$(29)	HOME UP (move cursor to upper left hand corner of display screen).
CHR$(30)	ERASE EOL (erase to end of line).
CHR$(31)	ERASE EOS (erase to end of screen).

In the statements shown below, X% is the column number (X-axis) and Y% is the line number (Y-axis). The upper left-hand corner is defined by the coordinates, (0,0). The following statement shows how direct cursor control (CHR$(14)) is used to position the cursor to (X%,Y%):

```
        PRINT #N%, CHR$(14%); CHR$(32%+Y%); CHR$(32%+X%);
```

The semicolons prevent automatic carriage return.

The statement shown below combines CHR$(14) and CHR$(32). The cursor is positioned to (X%,Y%) and the screen is cleared.

```
        PRINT #N%, CHR$(14%); CHR$(32$+Y%); CHR$(32%+X%); CHR$(31%);
```

4.6 VT50 CURSOR CONTROL AND SPECIAL CHARACTERS

The Full Duplex/Full Duplex with Local Copy switch on the VT50 specifies the source of data received in Remote mode. When this switch is set to Full Duplex with Local Copy, data entered at the keyboard is transmitted to the terminal's receiver at the same time it is sent to the host computer. When the switch is set to Full Duplex, data transmitted from the keyboard goes to the computer only. This data may be echoed from the computer back to the terminal to give a visual image of the information transmitted.

Because it permits an interactive relationship between the VT50 and the RSTS-11 system, Full Duplex is the most versatile of the two settings. For example, in Full Duplex the user can create unique escape sequences; the receipt of ESC 1 (27 49) from the terminal can call one routine, ESC 2 another, and so on.

The VT50 alphanumeric display terminal recognizes certain control characters that are not used on other terminals. These are summarized in Table 4-2.

Table 4-2
VT50 Special Display Characters

Character String	Character	Meaning
CHR$(8)		BACKSPACE (move cursor left one character).
CHR$(11)		CURSOR DOWN (one line, same position).
ESC (CHR$(155)) Followed By:		
CHR$(65)	A	CURSOR UP (one line, same position).
CHR$(67)	C	CURSOR RIGHT (one character).
CHR$(72)	H	HOME UP (moves cursor to upper left hand corner of display screen).
CHR$(74)	J	ERASE EOS (erase to end of screen).
CHR$(75)	K	ERASE EOL (erase to end of line).
CHR$(90)	Z	Requests the terminal to identify itself. The VT50 terminal will respond with ESC (27 47 65). The VT50 terminal with copier will respond with ESC (27 47 66).
CHR$(91)	[	Enables Hold Screen Mode.
CHR$(92)	\	Disables Hold Screen Mode.

4.7 PSEUDO KEYBOARD OPERATIONS

A pseudo keyboard is a non-physical device which allows one job to control other jobs on the system. The number of pseudo keyboards on the system is determined at system generation time. The system denotes a pseudo keyboard by a device designator of PK and associates each pseudo keyboard unit to a keyboard unit number but not to a physical terminal device.

The pseudo keyboard device has the characteristics of a terminal but has no physical device associated with it. As such, the pseudo keyboard has both input and output buffers to which user programs send input and from which they extract output. The output buffer of the pseudo keyboard is the input buffer for the keyboard unit and vice versa, yet no physical device is involved.

The system transfers data to the pseudo device in full duplex mode. This mode means that strings transmitted by PUT statements are echoed by the system and available to the program by GET, INPUT or INPUT LINE statements. In addition, a CR character (CHR$(13)) sent to the PK buffer is returned from the KB buffer as a CR and LF character sequence.

Use of a pseudo keyboard involves a controlling job and a controlled job. The controlling job initiates operations by performing output operations to the pseudo keyboard unit. It utilizes the pseudo keyboard to perform output to and receive input from the controlled job. The controlled job performs input/output operations on its own terminal, KB:. In effect, the controlled job does not know it is using a pseudo keyboard; only the controlling job is specifically using a pseudo keyboard.

A controlling job accesses a pseudo keyboard unit by the OPEN statement with the device designator PKn:, where n is the unit number of the pseudo keyboard. For example,

```
        10  OPEN "PK0:" AS FILE 1%
```

The OPEN statement associates pseudo keyboard unit 0 with internal channel number 1. If the device is assigned to or opened by another job, the ACCOUNT OR DEVICE IN USE error (ERR = 3) occurs. Otherwise the program can subsequently perform GET and PUT operations on the device by means of the buffer on channel 1.

To obtain data output from the controlled program, the controlling program executes a Record I/O GET statement on the proper internal channel. For example,

```
        40  GET #1
```

makes data from the pseudo keyboard available in the buffer for channel 1 of the controlling program. If no input is available, an END OF FILE ON DEVICE error (ERR = 11) occurs. When no error occurs, the BASIC-PLUS variable RECOUNT contains the number of bytes in the buffer. The response from a GET statement is immediate. The controlling program is never stalled pending data availability. This means that when a GET is executed, the contents of the buffer is returned to the controlling job - whether it is a single message, multiple messages, or a message fragment.

If the controlled job performs output faster than the controlling job can execute GET statements, the keyboard output buffer fills. Consequently, the controlled job enters an output wait state (TT) as if it were waiting for a real terminal. When the stall occurs, the system makes the controlling job eligible to run if it was in the SLEEP state so that it can execute GET statements and receive the output from the controlled job.

To perform output to a pseudo keyboard, the controlling program executes a Record I/O PUT statement with a coded value in the RECORD option. For example,

```
100  PUT #N%, RECORD R%, COUNT C%
```

The value N% is the internal channel number on which the PK device is open. The value C% is the number of bytes the program sends from the buffer to the PK device. The data sent from the buffer must be as one would type it at a keyboard. For example, if the controlling job sends a line which would normally be terminated by the RETURN key, the line sent must include only the CR character (CHR$(13)) and the value C% must reflect the CR character in the total number of bytes sent. The system automaticaly appends a LF character to a line terminated by a CR character. (The user normally does not type the LINE FEED key when he enters a line by typing the RETURN key.)

The value R% in the RECORD option determines the actions the system performs for the specific PUT statement. R% is a decimal integer whose value the system interprets on a bit by bit basis. The system examines only the low order four bits numbered 0 through 3 counting from right to left in the 16-bit word. The system executes the PUT statement depending upon whether certain bits in the value R% are on or off.

The flowchart of Figure 4-1 shows the actions the system performs by testing each bit in the RECORD R% option. For example, bit 0 (value =1) determines whether the system attempts to send data or performs tests before it sends data. Bit 1 (value = 2) is used to indicate whether the pseudo keyboard is waiting for a system command or is waiting for program input. Bit 2 (value = 4) actually sends data to the pseudo keyboard (if bit 0 is one) or simply returns control to the user program. Lastly, bit 3 (value = 8) tells the system, upon encountering a lack of small buffers, either to return an error or to wait until small buffers are available.

By alternately using PUT and GET statements, the controlling program starts a controlled job on the pseudo keyboard device. RECORD 1% sends data to the keyboard and can send LOGIN system program commands. A GET statement can retrieve output from the controlled job. For example, in response to the string "HELLO 10/10"+CHR$(13) sent to the pseudo keyboard, the system runs the LOGIN program. A subsequent GET statement can retrieve the echo and the PASSWORD: message printed by LOGIN. The controlling program can then send the proper password string and ensure that LOGIN accepted it.

Once the controlled job is running, the controlling program can send system commands, program commands, and program responses to the PK device using various RECORD option values in the PUT statement. The program should send only one line at a time and retrieve each program or system response. Also, the controlling program must not send a line unless the PK device is waiting for keyboard input. The controlling program should always check the PK device status before attempting to send data.

The RECORD 6% option (values 4 and 2) in a PUT statement can ensure that the controlled job is in editor mode (BASIC-PLUS command level). If it is waiting for KB input but is not in editor mode, the system generates error number 28. The user program must then force a CTRL/C combination to the controlled job. Otherwise, control is returned to the user program which can then transmit a system command.

Now to execute a program under the controlled job, the controlling program sends the RUN command with the program name to the PK device. To ensure that the controlled job is in the KB state awaiting keyboard input, the program first uses the RECORD 6% option in the PUT statement. If the controlled job is waiting for input, control is returned to the user program. Otherwise, the system generates error number 3.

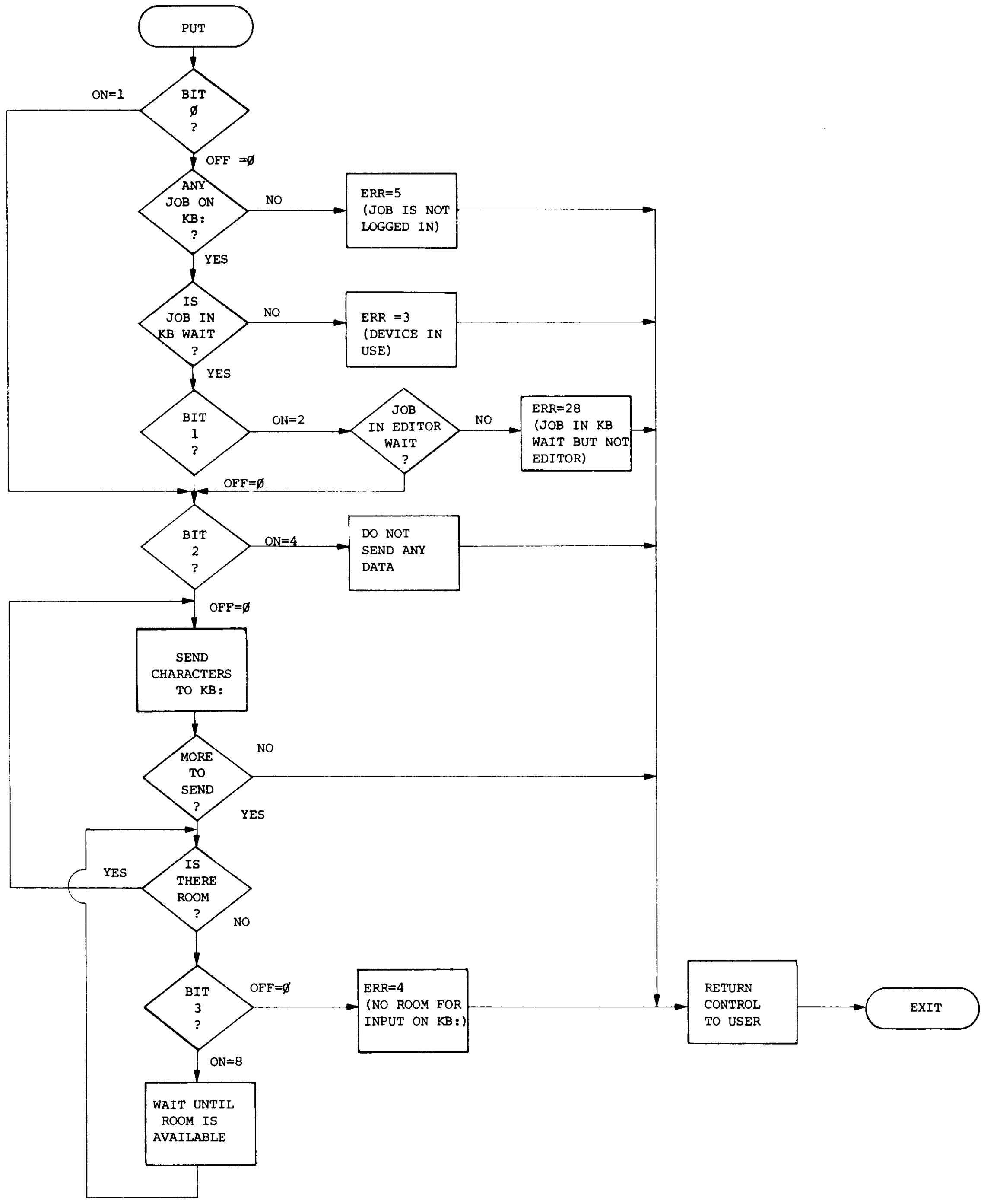


Figure 4-1
PUT Statement Actions for PK Output

CHAPTER 5

NON-FILE STRUCTURED DECTAPE

In non-file structured processing of DECtapes, the user program can access specific physical blocks on the DECtape. To initiate non-file structured processing, the user program gives only a device designator in the OPEN statement. Of the three conventional forms of the OPEN statement, OPEN FOR INPUT, OPEN, and OPEN FOR OUTPUT, only two are valid. The following two statements,

```
100  OPEN "DT1:" FOR INPUT AS FILE 1%
```

and

```
100  OPEN "DT1:" AS FILE 1%
```

are equivalent because both reading and writing of physical blocks on the device are permitted. The following statement:

```
100  OPEN "DT1:" FOR OUTPUT AS FILE 3%
```

is invalid since it attempts to create a file.

After opening a DECtape device for non-file structured processing, GET and PUT statements can retrieve and write specific physical blocks on the device by means of the RECORD option. The record number specified is interpreted as a block number. When the RECORD option specifies a negative block number, the designated block is accessed backwards. (Block 0 cannot be accessed backwards.) For example:

```
200 GET, #1% RECORD -4%
```

reads block 4 of the file opened on channel 1% backwards.

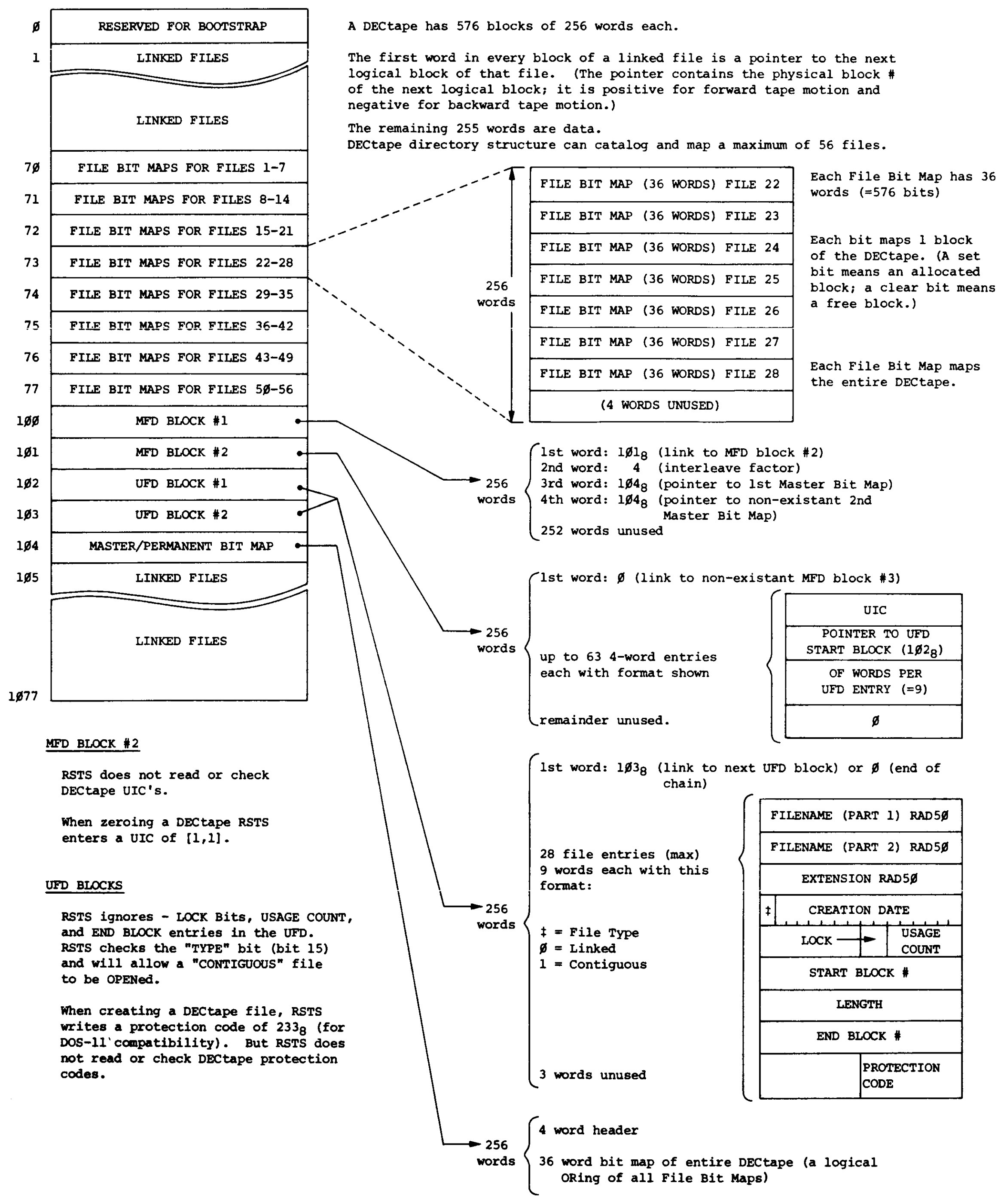
In writing non-file structured DECtape files, the user can specify how blocks should be accessed. But a file on a file structured DECtape is written on every fourth block (i.e., Block N, N+4, N+8, etc.) of the DECtape by the RSTS/E system. This procedure optimizes DECtape access time. When the system reaches the last block of the tape, it begins to write blocks backwards in intervals of four. It then repeats the entire process to fill in the available blocks on the DECtape. Therefore, to read a file structured DECtape in non-file structured mode, read every fourth block of the tape and use the RECORD -N% option to read alternate blocks (in the order in which they were written) backwards. Repeat this procedure to read the remaining blocks on the DECtape.

In file structured mode, since the blocks are not contiguous, the first word of each block of a file is a pointer to the next logical block of the file. These blocks are linked by these pointers. The DECtape format diagram (Figure 5-1) shown on the next page is provided so that non-file structured DECtape access time can be minimized.

A DECtape has 576 blocks of 256 words each.

The first word in every block of a linked file is a pointer to the next logical block of that file. (The pointer contains the physical block # of the next logical block; it is positive for forward tape motion and negative for backward tape motion.)

The remaining 255 words are data.
DECtape directory structure can catalog and map a maximum of 56 files.



Ø RESERVED FOR BOOTSTRAP
1 LINKED FILES
LINKED FILES
7Ø FILE BIT MAPS FOR FILES 1-7
71 FILE BIT MAPS FOR FILES 8-14
72 FILE BIT MAPS FOR FILES 15-21
73 FILE BIT MAPS FOR FILES 22-28
74 FILE BIT MAPS FOR FILES 29-35
75 FILE BIT MAPS FOR FILES 36-42
76 FILE BIT MAPS FOR FILES 43-49
77 FILE BIT MAPS FOR FILES 5Ø-56
1ØØ MFD BLOCK #1
1Ø1 MFD BLOCK #2
1Ø2 UFD BLOCK #1
1Ø3 UFD BLOCK #2
1Ø4 MASTER/PERMANENT BIT MAP
1Ø5 LINKED FILES
LINKED FILES
1Ø77
256 words
FILE BIT MAP (36 WORDS) FILE 22
FILE BIT MAP (36 WORDS) FILE 23
FILE BIT MAP (36 WORDS) FILE 24
FILE BIT MAP (36 WORDS) FILE 25
FILE BIT MAP (36 WORDS) FILE 26
FILE BIT MAP (36 WORDS) FILE 27
FILE BIT MAP (36 WORDS) FILE 28
(4 WORDS UNUSED)
Each File Bit Map has 36 words (=576 bits)
Each bit maps 1 block of the DECtape. (A set bit means an allocated block; a clear bit means a free block.)
Each File Bit Map maps the entire DECtape.
256 words
1st word: 1Ø1₈ (link to MFD block #2)
2nd word: 4 (interleave factor)
3rd word: 1Ø4₈ (pointer to 1st Master Bit Map)
4th word: 1Ø4₈ (pointer to non-existant 2nd Master Bit Map)
252 words unused
256 words
1st word: Ø (link to non-existant MFD block #3)
up to 63 4-word entries each with format shown
remainder unused.
UIC
POINTER TO UFD START BLOCK (1Ø2₈)
OF WORDS PER UFD ENTRY (=9)
Ø
256 words
1st word: 1Ø3₈ (link to next UFD block) or Ø (end of chain)
28 file entries (max) 9 words each with this format:
‡ = File Type
Ø = Linked
1 = Contiguous
3 words unused
FILENAME (PART 1) RAD5Ø
FILENAME (PART 2) RAD5Ø
EXTENSION RAD5Ø
‡ CREATION DATE
LOCK
USAGE COUNT
START BLOCK #
LENGTH
END BLOCK #
PROTECTION CODE
256 words
4 word header
36 word bit map of entire DECtape (a logical ORing of all File Bit Maps)

MFD BLOCK #2

RSTS does not read or check DECtape UIC's.

When zeroing a DECtape RSTS enters a UIC of [1,1].

UFD BLOCKS

RSTS ignores - LOCK Bits, USAGE COUNT, and END BLOCK entries in the UFD. RSTS checks the "TYPE" bit (bit 15) and will allow a "CONTIGUOUS" file to be OPENed.

When creating a DECtape file, RSTS writes a protection code of 233_8 (for DOS-11 compatibility). But RSTS does not read or check DECtape protection codes.

Figure 5-1
DECtape Format

CHAPTER 6

CARD READER

Standard (80-column) data processing cards can be read in any one of three modes: ASCII, packed Hollerith, or binary. One card can be read (and the information on it stored) in any mode.

6.1 ASCII MODE

The card reader reads cards punched with the standard ASCII codes, as shown in Appendix B. One of three sets of codes may be used: 029, 026, or 1401 EBCDIC. The code set for the system is specified during system generation. Cards punched in other formats are not acceptable to RSTS/E. The end-of-file card for RSTS/E contains a 12-11-0-1 or a 12-11-0-1-6-7-8-9 punch in card column 1. Reading an end-of-file card causes an END OF FILE ON DEVICE error (ERR = 11) to occur, which can be trapped with an ON ERROR GOTO statement.

The RECOUNT variable (see Section 12.3.1, BASIC-PLUS Language Manual) contains the number of characters read following every input operation. In the ASCII read mode, trailing spaces are ignored and carriage return and line feed characters are appended making the value of the RECOUNT variable two more than the number of punched columns per card. Consequently, the RECOUNT variable can have a value between 2 (for a blank card) and 82 (for 80 columns of data). For example, consider a card punched as follows:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

(columns 1 to 26 are punched, 27 through 80 are blank); the following program executes as shown:

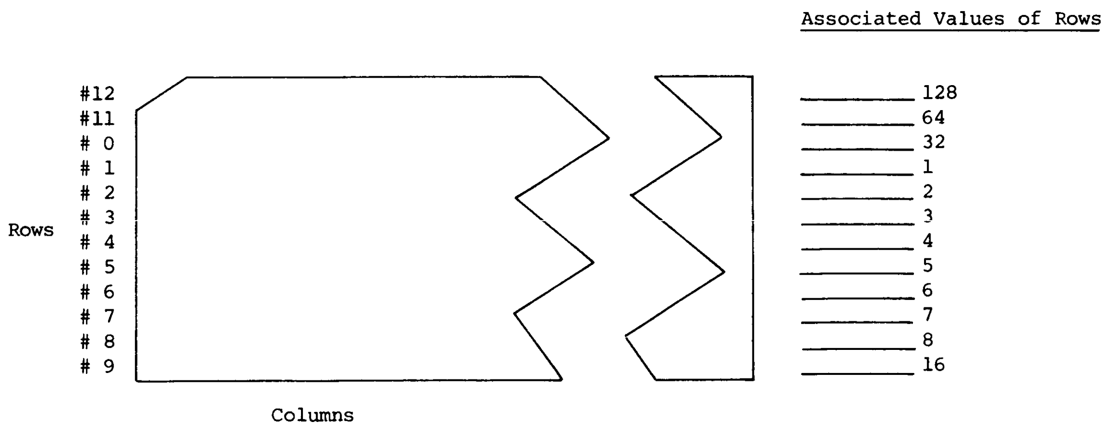
```
100  OPEN "CR:" AS FILE 1%
110  INPUT LINE #1%, A$
120  PRINT LEN(A$)
130  PRINT ">" A$ "<"
140  END

RUNNH

 28
>ABCDEFGHIJKLMNOPQRSTUVWXYZ
<
```

In this example the trailing spaces in card columns 27 through 80 are deleted, and the two characters, carriage return and line feed are added, making a total of 28 characters in the string A$.

Cards can be read with INPUT, INPUT LINE or GET statements. If a card is misread, or contains any illegal punches, a USER DATA ERROR ON DEVICE error occurs. If the card is read with a Record I/O GET statement, the buffer contains data for each column punched, and any columns that contain illegal punches are stored as RUBOUT (ASCII 127) codes. Therefore, the program can determine in which column(s) the error(s) occurred.

6.2 PACKED HOLLERITH MODE

In the packed Hollerith read mode, the value of the RECOUNT variable is always 80, since each of the 80 card columns corresponds to a single data byte and trailing spaces are not ignored. The value of each byte is the sum of the punched row positions, as shown in Figure 6-1.



BIT	7	6	5	4	3	2	1	0
ROW	12	11	∅	9	8		1-7	
VALUE	128	64	32	16	8	4	2	1

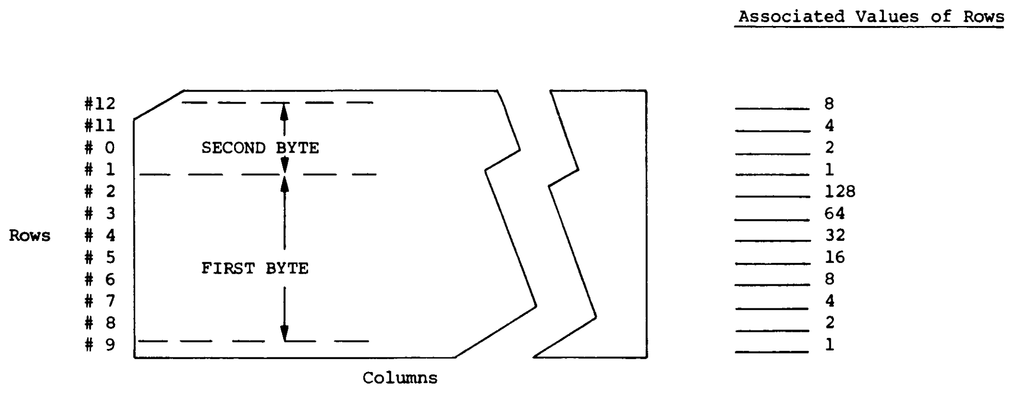
DIAGRAM #1

Figure 6-1
Packed Hollerith Read Mode

Notice that the associated values of rows 1 through 7 are simply 1 through 7 respectively. Only one of these seven rows can be punched per column. If none of these seven rows is punched, the value of the byte is 0.

6.3 BINARY MODE

The binary read mode associates two data bytes with each card column. Therefore, the value of the RECOUNT variable is always 160. Once again, the value of each byte is the sum of the punched row positions, as shown in Figure 6-2.



BIT	15	14	13	12	11	10	9	8	7	6	5	3	2	1	0
ROW	Ø	Ø	Ø	Ø	12	11	Ø	1	2	3	4	6	7	8	9
	SECOND BYTE								FIRST BYTE						

DIAGRAM #2

Figure 6-2
Binary Read Mode

6.4 SETTING READ MODES

A read mode is specified in an OPEN statement (with the MODE option) or a GET statement (with the RECORD option). The difference between specifying the read mode in a MODE option and in a RECORD option is discussed below. The corresponding values of the expressions in the MODE and RECORD options are listed in Table 6-3. The default mode is 0 (ASCII). When a MODE or RECORD option is used, the expression parameter must be specified; failure to do so results in an error message.

Table 6-3
Specifying Read Modes

	MODE or RECORD Expression	Specified Read Mode
OPEN Statement	MODE 0	ASCII
	MODE 1	Packed Hollerith
	MODE 2	Binary
GET Statement	RECORD 256	ASCII
	RECORD 257	Packed Hollerith
	RECORD 258	Binary

For example:

```
60  OPEN "CR:" FOR INPUT AS FILE 2%, MODE 1%
110 GET #2%, RECORD 258
```

Line 60, above, specifies the packed Hollerith read mode and line 110 specifies the binary read mode of operation for inputting the information on the first card.

A read mode specified in an OPEN statement supersedes previous read mode specifications. A read mode specified in a GET statement, however, overrides previous read mode specifications in the program for one card only. These concepts can best be illustrated by an example. Consider the program segment shown below.

```
                                                        Specified Read Mode
                                                           at This Point


100  OPEN "CR:" FOR INPUT AS FILE 1%, MODE 1%              Hollerith

200  GET #1%, RECORD 256%                                  ASCII

300  GET #1%                                               Hollerith
350  CLOSE 1%       ! OPTIONAL CLOSE IN THIS CASE
400  OPEN "CR:" FOR INPUT AS FILE 6%, MODE 0%              ASCII

500  GET #6%                                               ASCII

600  GET #6%, RECORD 258%                                  Binary

700  CLOSE 6%
```

After line 100, above, sets the read mode to Hollerith, line 200 overrides it, setting the read mode to ASCII temporarily. When line 300 is executed without a RECORD option, however, the read mode reverts to the OPEN mode -- in this case, Hollerith. The next OPEN statement (line 400) supersedes the previous one, setting the read mode to ASCII permanently. Line 500 is executed without a RECORD option, so the next card is read also in ASCII read mode. Closing a CR file (line 700), of course, cancels the card reader's read mode. When a file has been closed, executing an OPEN statement is the only way to re-establish a read mode.

CHAPTER 7

SYS SYSTEM FUNCTION CALLS AND THE PEEK FUNCTION

7.1 GENERAL SYS SYSTEM FUNCTION CALLS

SYS system function calls allow a user program to perform special I/O functions, to establish special characteristics for a job, to set terminal characteristics, and to cause the monitor to execute special operations.

The specified SYS format is employed for two reasons. One, the calls are unique to the BASIC-PLUS implementation of the BASIC language. As such, the calls are system dependent and have calling format different from any BASIC language call. Second, the SYS format allows the usage of a variable number of parameters.

SYS calls are separated into two classes: privileged and non-privileged. The privileged calls can be used only by a privileged user or by a privileged program. The non-privileged calls can be used by anyone and are completely safe in the sense that their misuse can do no damage to other programs.

7.1.1 SYS System Function Formats And Codes

The general format of the SYS call is as follows:

```
          V$ = SYS(CHR$(F) + O$)
```

where:

V$	is the target string returned by the call
F	is the SYS system function code
O$	is the optional (by function code) parameter string passed by the call

Function codes denoted by F in the general format are from zero through nine, inclusive. SYS calls which specify a code outside of these numbers or which pass a zero length string generate the ILLEGAL SYS() USAGE error (ERR = 18). The following list summarizes the codes and their usages. The subsequent pages describe the usage, calling format, and purpose of the calls.

Function Code(F)	Usage
0	Cancel ^O effect on console terminal
1	Enter tape mode on console terminal
2	Enable echoing on console terminal
3	Disable echoing on console terminal
4	Enable single character input mode (ODT submode) on console terminal
5	Exit to editor with no READY message
6	SYS call to the file processor
7	Get core common string
8	Put core common string
9	Exit to editor and set up NONAME program

7.1.2 General SYS System Function Calls

7.1.2.1 Cancel ^O Effect On Console Terminal - Not Privileged (F=0)

Data Passed:

Byte(s)	Meaning
1	CHR$(0%), the cancel ^O effect code
2	Optional. If specified, CHR$ (N%) where N% is the number of the channel on which the system executes the call.

Data Returned: The target string is equivalent to the passed string.

Discussion:

This call cancels the effect of the user typing a CTRL/O combination at the program's console terminal. If the call is in the form SYS(CHR$(0%) + CHR$(N%)), the system performs the action on the terminal open on channel N%. See Section 3.7 of the RSTS-11 System User's Guide for a description of the CTRL/O combination.

7.1.2.2 Enter Tape Mode On Console Terminal - Not Privileged (F=1)

Data Passed:

Byte(s)	Meaning
1	CHR$(1%), the enter tape mode code
2	Optional. If specified, CHR$(N%) where N% is the number of the channel in which the system executes the call.

Data Returned: The target string is equivalent to the passed string.

Discussion:

The action of this call is the same as that of the TAPE command described in the RSTS-11 System User's Guide. If the call is in the form SYS(CHR$(1%) + CHR$(N%)), the system performs the action on the terminal open on channel N%.

7.1.2.3 Enable Echoing On Console Terminal - Not Privileged (F=2)

Data Passed:

Byte(s)	Meaning
1	CHR$(2%), the enable echoing code
2	Optional. If specified, CHR$(N%) where N% is the number of the channel on which the system executes the call.

Data Returned: The target string is equivalent to the returned string.

Discussion:

This code cancels the effect of SYS calls with codes 1 and 3. If the form of the call is SYS(CHR$(2%) + CHR$(N%)), the action is performed on the terminal open on channel N%.

7.1.2.4 Disable Echoing On Console Terminal - Not Privileged (F=3)

Data Passed:

Byte(s)	Meaning
1	CHR$(3%), the disable echoing code
2	Optional. If specified, CHR$(N%) where N% is the number of the channel on which the system executes the call.

Data Returned: The target string is equivalent to the passed string.

Discussion:

This call prevents the system from echoing information typed at the console terminal. As a result, information such as a password is kept secret but accepted as valid input by the system. If the form of the call is SYS(CHR$(3%) + CHR$(N%)), the action is performed at the terminal open on channel N%.

7.1.2.5 Enable Single Character Input Mode (ODT Submode) On Console - Terminal - Not Privileged (F=4)

Data Passed:

Byte(s)	Meaning
1	CHR$(4%), the enable single character input mode code
2	Optional. If specified, CHR$(N%) where N% is the number of the channel on which the system executes the call.

Data Returned: The target string is equivalent to the passed string.

Discussion:

Allows a single character to be accepted as input from the terminal. Normally, the system waits until a line terminated by a RETURN, LINE FEED, FORM FEED, or ESCAPE character has been typed before accepting input. In single character mode, a character typed at the terminal is passed immediately to the program by the next keyboard input request statement without waiting for the delimiting character.

This function must be enabled prior to every input request statement that passes a single character to the program. A GET statement is used as the input request statement. (INPUT or INPUT LINE statements cause repeated generation of the input request until a line terminator is detected and, therefore, must not be used.)

If a program performs other lengthy operations before it executes either another SYS call and GET statement or other input/output operation at the terminal, it allows time for the user to type more than one character. To provide for such a possibility, the program should examine the system variable RECOUNT after executing each GET statement. This procedure determines how many characters the user typed between keyboard input operations and enables the program to process all the characters without losing any.

Since this function is used in the system program ODT.BAS, it is sometimes referred to as "ODT submode". If the form of the call is SYS(CHR$(4%) + CHR$(N%)), the action is performed on the terminal open on channel N%.

7.1.2.6 Exit To Editor With No READY Message - Not Privileged (F=5)

Data Passed:

Byte(s)	Meaning
1	CHR$(5%), the exit with no READY code

Data Returned: No data is returned.

Discussion:

This call causes the same effect as the END statement, except that it can appear anywhere in the program and does not cause a READY message to be printed.

7.1.2.7 FIP Function Call - Both Privileged and Not Privileged (F=6)

See Section 7.2 for a description of SYS calls to the file processor.

7.1.2.8 Get Core Common String - Not Privileged (F=7)

Data Passed:

Byte(s)	Meaning
1	CHR$(7%), the get core common string code

Data Returned: The target string is the contents of the core common area.

Discussion:

Allows a program to extract a single string from a data area loaded by another program previously run by the same job. The data area is called the core common and is from 0 to 127 8-bit bytes long. Refer to SYS function code 8.

7.1.2.9 Put Core Common String - Not Privileged (F=8)

Data Passed:

Byte(s)	Meaning
1	CHR$(8%), the put core common string code
2-128	The string to put in the core common area

Data Returned: The target string is the passed string.

Discussion:

Allows a program to load a single string in a common data area called core common. This string can be extracted later by another program, running under the same job and called via the CHAIN statement. The string is from 0 to 127 8-bit bytes long. If the string to be put into the core common area is longer than 127 bytes, the system sets the length of the core common string to 0. Refer to SYS function code 7.

This function provides a means for passing a limited amount of information when a CHAIN statement is executed. If a larger amount of information is to be passed, it must be written to a disk file and read back by the later program.

7.1.2.10 Exit To Editor And Set Up NONAME Program - Not Privileged (F=9)

Data Passed:

Byte(s)	Meaning
1	CHR$(9%), the exit and set up NONAME code

Data Returned: The target string is equivalent to the passed string.

Discussion:

This function causes the same actions as the END statement placed in the code, and, in addition, clears the program out of memory. This is the proper method of stopping a program that is not to be rerun. Also, the same action is performed by the command NEW NONAME.

7.2 SYS SYSTEM FUNCTION CALLS TO FIP (FUNCTION CODE 6)

The SYS function call whose code is 6 is a more specialized case of the general system function call. It is specialized by a subfunction code called the FIP code. The FIP code causes a dispatch call to be made to special resident or non-resident code that performs File Processing.

The format of the call is:

```
V$ = SYS(CHR$(6%) + CHR$(F0) + O$)
```

where:

V$	is the data (target) string returned by the call
F0	is the FIP function code
O$	is the optional (by function) parameter string

The general format of the target variable (V$) is:

Byte(s)	Meaning
1	Job number times 2
2	Value of Internal Function called (meaningless to general user)
3-30	Data returned

NOTE

Thirty bytes are always passed back. Unused bytes are either internal data or 0.

The proper use of the FIP system function call requires that the user program build a parameter string to pass and that the program later extract the data from the returned string, called the target string. Each call returns a string of 30 bytes, each byte (or character) of which may or may not contain useful information. The descriptions of the FIP codes specify the contents of each useful byte in the string, from which the user determines whether the information contained is of interest.

7.2.1 Building A Parameter String

Some FIP calls require no parameters except the function and subfunction codes; other FIP calls require either variable length parameter strings or very simple parameter strings. For such FIP calls, it is usually easiest to set up and execute the function call in a single statement. The following sample statements show the procedure.

```
A$ = SYS(CHR$(6%) + CHR$(-7%))
        !ENABLE CTRL/C TRAP
```

```
                !(NO PARAMETER STRING)

A$ = SYS(CHR$(6%) + CHR$(-10%) + "DK0:FILE.EXT")
                !FILE NAME STRING SCAN
                !(VARIABLE LENGTH
                !  PARAMETER STRING)

A$ = SYS(CHR$(6%) + CHR$(-8%) + CHR$(1%))
                !FCB/DDB INFORMATION
                ! FOR FILE OPEN ON
                ! CHANNEL 1
                ! (SIMPLE PARAMETER
                ! STRING)
```

Many FIP calls require more complex data formats. For example, the kill a job FIP call, F0 = 8, requires byte 3 to be the job number to kill, byte 27 to be 0, and byte 28 to be 255. A recommended method of building the complex parameter string to pass to a function is to dimension a 30 byte list (an integer array) and set the items in the list to values which map into those required in the parameter string format. The list can later be set to a character string by the CHANGE statement before it is passed as the parameter string of the FIP system function call. The resulting character string is in proper format and contains the correct byte values so that it can be placed as the parameter string of the FIP system function call. For example,

```
10  DIM A%(30%)                       !the string is 30 bytes
20  J% = 4%                           !kill job number 4
30  A%(I%) = 0% FOR I% = 0% TO 30%    !ZERO UNUSED ENTRIES
40  A%(0%) = 30%                      !0th element is length of
list
50  A%(1%) = 6%                       !SYS call code 6 (FIP call)
60  A%(2%) = 8%                       !FIP code 8 (kill job)
70  A%(3%) = J%                       !job number to kill
80  A%(27%) = 0%                      !this byte must be 0
90  A%(28%) = 255%                    !this byte must be 255
```

Following the code which builds the list is the CHANGE statement and the call itself, as shown below.

```
100  CHANGE A% TO A$          !generates character
           .                  !string from the
           .                  !integer list

200  B$ = SYS(A$)             !invoke system function call
```

7.2.2 Unpacking The Returned Data

In the example above, the action performed (kill a job) rather than the data returned is of importance. However, many FIP calls return a data string which is the primary interest of the user. In such a case, the data in the string must be unpacked.

As in the case of building the parameter string, there are two recommended methods of unpacking the returned string. If the user needs only a few pieces of the data, it may be easiest to operate directly on the returned string. For example, if the user wants only the 4-byte Radix-50 representation of a 6-byte string, the filename string scan FIP call (FIP code -10) can be used as follows:

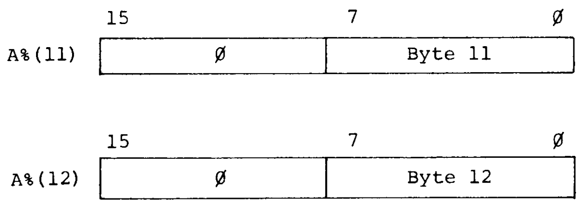
```
A$ = MID(SYS(CHR$(6%) + CHR$(-10%) + S$), 7%, 4%)
```

to extract bytes 7 through 10 of the returned string. To extract numeric data, ASCII or CVT$% functions can be used.

In some cases, many pieces of the returned data are needed. In other cases, the string returned by the FIP call is needed later to set up another FIP call. In such cases, the user program can transform the returned string to a 30-byte list using a CHANGE statement,

```
CHANGE A$ TO A%
          or
CHANGE SYS(...) TO A%
```

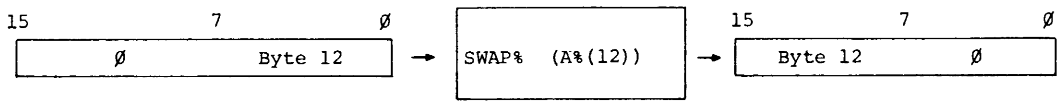
When the returned string has been converted in this manner, it is necessary to do further conversions in order to get numeric data into a usable form. Take, for example, the data returned by a FIP code of 15 (directory lookup on index). The layout of the data returned specifies that bytes 11 and 12 are the filename extension encoded in Radix-50 format. In order to convert those bytes into an ASCII string, to OPEN the file, for example, the RAD$ BASIC-PLUS function must be used on the two bytes converted to a single integer. The integer representation of each byte, however, occupies a full word, 16 bits in length. Thus, list items number 11 and 12 appear as the following:



A%(11) contains the low byte portion of the Radix-50 word; A%(12) contains the high byte portion of the Radix-50 word. The two bytes must be combined into a single word and converted to the proper character string representation. This is accomplished by the following:

```
S$ = RAD$(A%(11) + SWAP%(A%(12)))
```

The SWAP% function reverses the bytes (the low byte takes the high byte position and vice-versa) in an integer word. Graphically, the operation appears as follows:



Thus, byte 12 takes the high byte position in the word. The two words are then combined by the + operator to form one word. The RAD$ function performs the conversion on that one integer word to produce the 3-character string representation of the file name extension.

The character string is assigned to the character variable S$ and is in ASCII format.

To convert a longer string from Radix-50 to ASCII format, the above procedure must be applied to each two bytes in the string. For example, the filename from FIP call 15 is returned in bytes 7 through 10. In order to convert these bytes to ASCII format, use the following routine.

```
A$ = RAD$(A%(7%) + SWAP%(A%(8%))
B$ = RAD$(A%(9%) + SWAP%(A%(10%)
F$ = A$ + B$
```

or, in a single statement,

```
F$ = RAD$(A%(7%) + SWAP%(A%(8%))) +
     RAD$(A%(9%) + SWAP%(A%(10)))
```

7.2.3 Notation And References Used In FIP Call Descriptions

Many FIP calls require that a project-programmer number (PPN) be specified in the calling string, and several return a PPN. Where such is the case, the PPN field is in the following general form.

```
Bytes X-(X+1)        PPN
```

The value X is odd. The intended meaning of this notation is that byte X in the string holds the programmer number, and byte (X+1) holds the project number. For example, to set up a FIP call to zero an account on a disk (FIP code 13), the calling format shows

```
Bytes 5-6            Project-programmer number
```

If the call is to be set up in a 30-entry list, A%, then the format requires that

```
A%(5%) = programmer number
A%(6%) = project number
```

Many of the FIP calls described in this chapter return or require integer data in two (consecutive) bytes of the returned data string. When this is the case, the field in the returned string is described in the format:

```
Bytes X-(X+1)        integer value
```

If the return string is to be processed directly (that is, without changing it to an integer array), then the integer value of the two bytes can be obtained using the following statement.

```
I% = SWAP%(CVT$%(MID(A$,X,2%)))
```

where A$ holds the returned string. If the returned data string is first transferred to an integer array, A%, using the CHANGE statement, then the integer value can be obtained using the statement below.

```
I% = A%(X) + SWAP%(A%(X+1%))
```

For example, the Get Monitor Tables (Part I) FIP call (FIP code -3) returns the address of monitor's job table in bytes 11 and 12. If A$ holds the returned string, then either of the following two routines put the address of the job table into the integer variable I%.

```
I% = SWAP%(CVT$%(MID(A$,11%,2%)))

             or

CHANGE A$ TO A%
I% = A%(11%) + SWAP%(A%(12%))
```

In some integer fields in the FIP calls, the value is a full 16-bit unsigned integer between 0 and 65535. The sign bit indicates an extra power of two rather than positive or negative. Since an integer value in BASIC-PLUS is between -32768 and +32767; any value greater than 32767 must be stored as a floating point value. Assume that in some SYS call, an unsigned integer is returned in bytes 5 and 6, and that the returned string has been changed to an array, A%. As always, the high byte of the integer is in byte 6, the low byte in byte 5. The statement

```
          Q = 256.*A%(6%) + A%(5%)
```

gets the full 16 bit value into the floating point variable, Q. Q is always positive. Note that replacing the 256.* with SWAP%() causes the expression to be first evaluated as a normal integer expression, and then changed to a floating point value. The resulting value is between -32768 and +32767. The 256.* forces the expression to be evaluated as a floating point number.

To convert an unsigned integer to two bytes to pass to a SYS call also requires special processing. Assuming that Q holds the unsigned value, and that the value is to be placed in A%(5%) (low order) and A%(6%) (high order), then the most direct method of transformation is:

```
          A%(6%) = Q/256.
          A%(5%) = Q-A%(6%)*256.
```

On PDP-11 computers without FIS or FPP (floating-point hardware), division operations are relatively slow. On these machines, a faster method is:

```
          10  IF Q<32768.THEN Q% = Q
              ELSE IF Q = 32768.  THEN Q% = 32767% + 1%
              ELSE    Q = Q-65536.
          20  A%(5%) = Q% AND 255%
          30  A%(6%) = SWAP% (Q%) AND 255%
```

The disadvantage of this second method is that it requires more code.

The filename string scan SYS function (FIP code -10) is useful as a "front-end" for many FIP functions. Most of the FIP calls which require device or filename information in their parameter strings expect information in the format which the FIP -10 call returns it. For example, FIP code 17, lookup a file by name, expects its calling string to be passed in exactly the same format as that returned by the FIP -10 call, with a change of only four data bytes. The following routine sets up and executes the lookup call on the file DK0:INVENT.DAT[10,20] using the filename string scan FIP call.

```
10 DIM A%(30%)                                  ! SET UP LIST ARRAY.
20000 A$="DK0:INVENT.DAT[10,20]"                ! SET UP VARIABLE.
20010 CHANGE SYS(CHR$(6%)+CHR$(-10%)+A$) TO A%
                                                ! DO THE FNS CALL TO
                                                ! SET UP ARRAY PROPERLY.
20020 A%(0%)=30%                                ! SET UP FOR 30 LONG.
20030 A%(1%)=6%                                 ! THIS IS A FIP CALL
20040 A%(2%)=17%                                ! DISK LOOKUP BY NAME.
20050 A%(3%),A%(4%)=0%                          ! SET INDEX TO ZERO.
20060 CHANGE A% TO A$                           ! SET UP AS A STRING
20070 CHANGE SYS(A$) TO A%                      ! AND DO THE CALL.
32767 END
```

In order to avoid redundancy in the descriptions in Section 7.2, any field for any of the calls which are either passed to or returned from the function in the same format as that returned by FIP code -10 are identified by a + superscript after the field specification. For a detailed explanation of fields so identified, see Section 7.2.4.1.

Table 7-1 is a quick reference index of the FIP functions in order of FIP code (F0). For detailed information on each of the functions, refer to the Section shown beside the name in the table.

This page left blank intentionally.

Table 7-1
FIP SYS Calls (by Sub-function Code)

FUNCTION CODE (F0)	PRIVILEGED STATUS	FUNCTION NAME	PAGE
-23	No	Terminating file name string scan	7-20
-22	Yes	Set special run priority	7-56
-21	Yes	Drop (temporary) privileges	7-59
-20	Yes	Lock/unlock job in core	7-58
-19	Yes	Set number of logins	7-89
-18	Yes	Add run-time system	7-111
	Yes	Remove run-time system	7-113
	Yes	Load run-time system	7-114
	Yes	Unload run-time system	7-116
-17	Yes	Name run-time system	7-110
-16	Yes	System shutdown	7-38
-15	Yes	Accounting dump	7-93
-14	Yes	Change system date/time	7-39
-13	Yes	Change priority/run burst/job size	7-54
-12	No	Get monitor tables - Part II	7-107
-11	Yes	Change file backup statistics	7-70
-10	No	Filename string scan	7-20
-9	Yes	Hangup a dataset	7-40
-8	No	FCB/DDB Information	7-108
-7	No	CTRL/C Trap enable	7-36
-6	Yes(1)	Poke core	7-88
-5	Yes	Broadcast to terminal	7-41
-4	Yes	Force input to terminal	7-42
-3	No	Get monitor tables - Part I	7-105
-2	Yes	Disable logins	7-43
-1	Yes	Enable logins	7-44
0	Yes	Create user account	7-60
1	Yes	Delete user account	7-62
2	Yes	Clean up a disk pack	7-48
3	Yes	Disk packs	7-45
	Yes	TTYSET	7-64
4	Yes	Login	7-72
5	Yes	Logout	7-74
6	Yes	Attach	7-77
	Yes	Reattach	7-79
7	Yes	Detach	7-75
8	Yes	Change password/quota	7-49
	Yes	Kill job	7-51
	Yes	Disable TTY	7-52
9	No	Return error messages	7-29
10	No	Assign/reassign device	7-30
11	No	Deassign device	7-32

----

(1)Poke core can be executed only from account [1,1].

Table 7-1 (Cont.)
FIP SYS Calls (by Sub-function Code)

FUNCTION CODE (F0)	PRIVILEGED STATUS	FUNCTION NAME	PAGE
12	No	Deassign all devices	7-33
13	Both	Zero a device	7-34
14	Both	Read or Read and Reset Accounting Data	7-90
15	No	Directory lookup on index	7-95
	No	Special magtape directory lookup	7-97
16	Yes	Set terminal characteristics	7-64
17	No	Disk directory lookup on filename	7-100
	No	Disk wildcard directory lookup	7-101
18	No	Send a message	7-85
	Yes	Declaring a receiver and receiving a message	7-82
	Yes	Remove from receive table	7-87
19	Yes	Enable/disable disk cacheing	7-109
--	Yes	PEEK function	7-117

The privileged status column indicates whether the SYS call can be used only by a privileged user or by any user. A non-privileged user who attempts to call a privileged SYS function always receives the ILLEGAL SYS() USAGE error (ERR = 18). The notation BOTH in the privileged status column indicates that some facilities of the specified function are available to a non-privileged user, while the privileged user has a more powerful set.

7.2.4 General Utility SYS Calls To FIP

The SYS calls to the file processor described in this section are available to both privileged and non-privileged users.

7.2.4.1 File Name String Scan - Not Privileged (F0=-10) (F0=-23)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-10), the filename string scan code. CHR$(-23) is the same as CHR$(-10) except that the scan terminates on certain characters. See discussion.
3-?	Character string to scan; can be any length.

Data Returned: Sets the STATUS variable and returns the following.

Byte(s)	Meaning
1-4	Internal coding
5-6	Project-programmer number (0 means the current account)
7-10	File name in Radix-50 format
11-12	Filename extension in Radix-50 format
13-20	Not used
21	If no protection code is found, this byte is 0. Otherwise, this byte is 255 and byte 22 contains the protection
22	Protection code when byte 21 is 255
23-24	Device name in ASCII format or 0 if none is found. For example, DK and DT
25	Device unit number if byte 26 is 255
26	If this byte is 0, no explicit unit number was found for the device. If this byte is 255, the value in byte 25 is the explicitly specified device unit number. The 255 value here indicates that a zero in byte 25 is explicitly unit 0 of the device.

Byte(s)	Meaning
27-28	First flag word. See discussion.
29-30	Second flag word. See discussion.

Possible Errors:

Meaning	ERR Value
ILLEGAL FILE NAME The character string scanned contains unacceptable characters. See Section 9.1 of the BASIC-PLUS Language Manual for a description of a file specification.	2

Discussion:

The file name string scan function determines specific file syntax information (for example, whether a given file name is valid) and returns information in the format required for all other file and device related SYS calls. In addition, it provides a means of packing a string in Radix-50 format.

The STATUS variable (see Section 12.3.5 of the BASIC-PLUS Language Manual) is set for the device type found in the string scanned.

The following example demonstrates how a string can be converted to Radix-50 format by a user defined function and the file name string scan SYS call.

```
10  DEF FNPO$(A$) = MID (SYS(CHR$(6%)+
      CHR$(-10%)+A$),7%,4%)
      ! PACK 6 CHARACTERS TO RADIX-50
```

The function FNPO$ returns a 4-character string which is the Radix-50 representation of the first six characters of A$. (Note that no error handling is included and that errors can occur.) The file name string scan call is the only function in BASIC-PLUS which packs a string in Radix-50 format. To pack strings longer than six characters, the user must make multiple calls to the SYS function.

The two words in bytes 27 and 28 and in bytes 29 and 30 hold easily accessible flags indicating exactly what fields in the source string were found and what kind of information they contained. For the purposes of the discussion, it is assumed that the returned string was converted by a CHANGE statement to an integer array, M%(30%). The flag words are then created by doing the proper arithmetic operations on the bytes, as shown:

```
flag word 1:  S0% = M%(27%)+SWAP%(M%(28%))
flag word 2:  S1% = M%(29%)+SWAP%(M%(30%))
```

Once these two words are created, the data coded into them is accessible by means of an AND operation between the word and the bit relating to a particular piece of information (see Section 2.8.7 for information about the AND operation). Each bit of the PDP-11 word can be used to hold a YES or NO answer. In the case of S0%, only the high-order 8 bits are used. In the case of S1%, all 16 bits are used.

Flag word 1 is redundant; that is, all information returned in flag word 1 is also in flag word 2. Flag word 2 holds much more information than flag word 1.

In the following discussion, it is assumed that bytes 27 and 28 have been put into S0% and bytes 29 and 30 have been put into S1% as described above.

Flag word 1: where S0% = M%(27%)+SWAP%(M%(28%))

Bit	Comparison	Meaning
8	S0 AND 256%<>0%	Filename was found in the source string (and is returned in Radix-50 format in bytes 7 through 10)
9	S0% AND 256% = 0%	No filename found
	S0% AND 512%<>0%	A dot was found in source string
	S0% AND 512% = 0%	No dot was found in source string implying that no extenson could have been specified either
10	S0% AND 1024%<>0%	A project-programmer number was found in source string
	S0% AND 1024% = 0%	No project-programmer number was found
11	S0% AND 2048%<>0%	A left angle bracket (<) was found in source string implying that a protection code was found
	S0% AND 2048% = 0%	No left angle bracket (<) was found (no protection was specified)
12	S0% AND 4096%<>0%	A colon (but not necessarily a device name) was found
	S0% AND 4096% = 0%	No colon was found implying that no device could have been specified
13	S0% AND 8192%<>0%	Device name was specified and specified device name was a logical device name
	S0% AND 8192% = 0%	Device name (if specified) was an absolute (non-logical) device name (if device name was not specified, this will be 0)
15	S0%<0%	Source string contained wild card characters (either ? or * or both) in filename, extension or project-programmer number fields. In addition, the device name specified, though a valid logical device name, possibly does not correspond to any of the logical device assignments currently in effect. The user program must extract the device name and attempt to access the unit. (See Section 7.2.16.4 for a description of wild card file specifications.)

Flag Word 2: where S1% = M%(29%)+SWAP%(M%(30%))

Bit	Comparison	Meaning
0	S1% AND 1%<>0%	File name was found in the source string
	S1% AND 1% = 0%	No file name was found (and the following two comparisons return 0)
1	S1% AND 2%<>0%	File name was an * character and is returned in bytes 7 through 10 as the Radix-50 representation of the string "??????".
	S1% AND 2% = 0%	File name was not an * character
2	S1% AND 4%<>0%	Filename contained at least one ? character
	S1% AND 4% = 0%	Filename did not contain any ? characters
3	S1% AND 8%<>0%	A dot (.) was found
	S1% AND 8% = 0%	No dot was found implying that no extension was specified (and the following three comparisons return 0)
4	S1% AND 16%<>0%	An extension was found (that is, the field after the dot was not null)
	S1% AND 16% = 0%	No extension was found (the field after the dot was null - the following two comparisons return 0)
5	S1% AND 32%<>0%	Extension was an * character and is returned in bytes 11 and 12 as the Radix-50 representation of the string "???"
	S1% AND 32% = 0%	Extension was not an * character
6	S1% AND 64%<>0%	Extension contained at least one ? character
	S1% AND 64% = 0%	Extension did not contain any ? characters
7	S1% AND 128%<>0%	A project-programmer number was found
	S1% AND 128% = 0%	No project-programmer number was found (the following two comparisons return 0)

8	S1% AND 256%<>0%	Project number was an * character (that is the projelct-programmer number was of the form [*,PROG]) and is returned in byte 6 as 255
	S1% AND 256% = 0%	Project number was not an * character
9	S1% AND 512%<>0%(1)	Programmer number was an * character (that is, the project-programmer number was of the form [PROJ,*] and is returned in byte 5 as 255
	S1% AND 512% = 0%	Programmer number was not an * character
10	S1% AND 1024%<>0%	A protection code was found
	S1% AND 1024% = 0%	No protection code was found
11	S1% AND 2048%<>0%	The protection code currently set as default by the current job was used
	S1% AND 2048% = 0%	The assignable protection code was not used (protection code given is either the system default, 60, or that found in the source string)
12	S1% AND 4096%<>0%	A colon (:), but not necessarily a device name, was found in the source string
	S1% AND 4096% = 0%	No colon was found (no device could have been specified); the following three comparisons return 0
13	S1% AND 8192%<>0%	A device name was found
	S1% AND 8192% = 0%	No device name was found; the following two comparisons return 0
14	S1% AND 16384%<>0%	Device name specified was a logical device name
	S1% AND 16384% = 0%	Device name specified was an actual device name
15	S1%<0%	The device name specified was logical and is not assigned to some actual device; the logical name is returned in bytes 23 through 26 as a Radix-50 string.

----

(1)Note that if the project-programmer number was of the form [*,*], then both bit 8 and bit 9 of the data byte returned are non-zero values.

S1%>=0%	The device name specified, if any, was either an actual device name, or a logical device name to which a value has been assigned; the physical device name is returned in bytes 23 through 26 as described above

Since flag word 2 contains all the information returned in flag word 1 plus more information, it is the more useful of the two words. The following program uses this word. It prints out a list of all the bits returned in the word.

```
5 DIM M%(30%)                                   ! SET UP AN ARRAY TO RETURN TO
10 PRINT "STRING TO SCAN";
20 INPUT LINE S$
30 S$=CVT$$(S$,-1%)                             ! GET RID OF GARBAGE BYTES
40 CHANGE SYS(CHR$(6%)+CHR$(-10%)+S$) TO M%
50 S1%=M%(29%)+SWAP%(M%(30%))
100 IF S1% AND 1%        THEN    PRINT "FILENAME FOUND"
110 IF S1% AND 2%        THEN    PRINT "FILENAME WAS AN *"
120 IF S1% AND 4%        THEN    PRINT "FILENAME HAD '?'S"
130 IF S1% AND 8%        THEN    PRINT "DOT (.) FOUND"
140 IF S1% AND 16%       THEN    PRINT "NON-NULL EXTENSION FOUND"
150 IF S1% AND 32%       THEN    PRINT "EXTENSION WAS '*'"
160 IF S1% AND 64%       THEN    PRINT "EXTENSION HAD '?'S"
170 IF S1% AND 128%      THEN    PRINT "PPN FOUND"
180 IF S1% AND 256%      THEN    PRINT "PROJECT NUMBER WAS '*'"
190 IF S1% AND 512%      THEN    PRINT "PROGRAMMER NUMBER WAS '*'"
200 IF S1% AND 1024%     THEN    PRINT "PROTECTION CODE FOUND"
210 IF S1% AND 2048%     THEN    PRINT "ASSIGN'D PROTECTION USED"
220 IF S1% AND 4096%     THEN    PRINT "COLON (:) FOUND"
230 IF S1% AND 8192%     THEN    PRINT "DEVICE NAME FOUND"
240 IF S1% AND 16384%    THEN    PRINT "DEVICE NAME WAS LOGICAL"
250 IF S1%<0%            THEN    PRINT "DEVICE NAME WAS NOT ASSIGN'D"
260 IF S1% AND 4096%     THEN
        IF S1%>0%        THEN    PRINT "'STATUS' HAS BEEN SET"
490 PRINT FOR I%=1% TO 2%
500 GOTO 10
32767 END
```

The following examples show some of the above messages:

```
STRING TO SCAN? ABCDEF.EXT
FILENAME FOUND
DOT (.) FOUND
NON-NULL EXTENSION FOUND


STRING TO SCAN? SY:FILENM.DEX
FILENAME FOUND
DOT (.) FOUND
NON-NULL EXTENSION FOUND
COLON (:) FOUND
DEVICE NAME FOUND
'STATUS' HAS BEEN SET
```

```
STRING TO SCAN? SY:FILENM.EXT[1,203]
FILENAME FOUND
DOT (.) FOUND
NON-NULL EXTENSION FOUND
PPN FOUND
COLON (:) FOUND
DEVICE NAME FOUND
'STATUS' HAS BEEN SET


STRING TO SCAN? SY:FILENM.EXT[2,103]<52>
FILENAME FOUND
DOT (.) FOUND
NON-NULL EXTENSION FOUND
PPN FOUND
PROTECTION CODE FOUND
COLON (:) FOUND
DEVICE NAME FOUND
'STATUS' HAS BEEN SET


STRING TO SCAN? SY:FILENNM.EXT[*,201]
FILENAME FOUND
DOT (.) FOUND
NON-NULL EXTENSION FOUND
PPN FOUND
PROJECT NUMBER WAS '*'
COLON (:) FOUND
DEVICE NAME FOUND
'STATUS' HAS BEEN SET


STRING TO SCAN? SY:A.*
FILENAME FOUND
DOT (.) FOUND
NON-NULL EXTENSION FOUND
EXTENSION WAS '*'
COLON (:) FOUND
DEVICE NAME FOUND
'STATUS' HAS BEEN SET


STRING TO SCAN?

STRING TO SCAN? SY:FILE??.EXT
FILENAME FOUND
FILENAME HAD '?'S
DOT (.) FOUND
NON-NULL EXTENSION FOUND
COLON (:) FOUND
DEVICE NAME FOUND
'STATUS' HAS BEEN SET


STRING TO SCAN? :A
FILENAME FOUND
COLON (:) FOUND
'STATUS' HAS BEEN SET
```

The STATUS variable is set or not set depending on the presence or absence of a device in the string scanned. The following three conditions pertain.

a. When no device name is found in the string (no colon is found), the STATUS is random. This condition pertains when bit 12 of flag word 2 tests as equal to 0.

b. When the device name is logical and untranslatable (an actual device is not assigned), STATUS is random. This condition pertains when bits 12, 13 and 14 of flag word 2 test as not equal to 0 and bit 15 tests as on (S1%<0%).

c. When the device name is either an actual device name or is logical and translatable (an actual device is assigned), STATUS is set for the device. This condition pertains when bit 12 tests as not equal to 0 and bit 15 tests as equal to 0 (S1%>=0%).

Line 260 of the sample program shows the test to determine when STATUS is set by the call.

The file name string scan using CHR$(-23%) in place of CHR$(-10%) terminates without error on the following characters.

=	(equality sign)
/	(slant)
;	(semi-colon)
,	(comma)
end of string	

The number of unscanned characters is returned in the BASIC-PLUS variable RECOUNT. For example,

```
S$=SYS(CHR$(6%) + CHR$(-23%) + "SY:[1,4]ABC<40>")
```

returns the data as described above for CHR$(-10%) and RECOUNT equals 0. The following call

```
S$ = SYS(CHR$(6%) + CHR$(-23%) + "SY:[1,4]ABC<40>,DT:DEF")
```

returns the data described above for the string "SY:[1,4]ABC<40>" with RECOUNT equals 7. (The scan terminates on the comma between file names.)

Any other characters, including the angle bracket character (<), generate an error and none of the data is returned. Thus, the call with CHR$(-23%) simplifies the task of decoding command strings with multiple file names.

7.2.4.2 Return Error Message - Not Privileged (F0=9)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(9), the return error message code
3	CHR$(E%), where E% is the RSTS ERR variable number and is between 0 and 127
4-30	Not used

Data Returned:

Byte(s)	Meaning
1	The current job number times 2
2	The current keyboard number times 2
3-30	Error message padded to length 28 with CHR$(0%) characters.

Possible Errors: No errors are possible.

Discussion:

This SYS system function call extracts error message text from the file ERR.SYS stored under account [0,1] or in the RSTS.CIL file if ERR.SYS does not exist. The text is associated with the value of the ERR variable passed as byte 3 of the call. The number in byte 2 of the returned string is two times the number of the keyboard involved in generating the error. This is an exception to the conventional contents of byte 2 which usually contains the job number times 2. A sample usage of the call is to print the system header line containing the system name and the local installation name. To do this, the character representation of the ERR value of 0% is used in the call.

```
10 INPUT "ERROR NUMBER";E%
20 S$=SYS(CHR$(6%)+CHR$(9%)+CHR$(E%))
30 S1$=MID(S$,3%,INSTR(3%,S$,CHR$(0%))-3%)
40 PRINT S1$
50 PRINT FOR I%=1% TO 2%
60 GOTO 10
32767  END
```

To extract the message text from the data returned by the SYS call, the program executes an INSTR function based on the NUL byte (FILL character) indicating the end of the text. The MID substring of the returned data string, starting at byte number 3, extracts the number of bytes according to the value returned by the INSTR function.

Error numbers used in the call can include those associated with recoverable and non-recoverable errors.

```
RUNNH
ERROR NUMBER?  0
RSTS V06A-01 SYSTEM #880
```

7.2.4.3 Assign/Reassign Device - Not Privileged (F0=10)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(10%), the assign/reassign device code
3-6	Not used
7	Must be 0 for assign; for reassign, must be the job number to reassign the device to
8	Must be 0
9-22	Not used
23-24+	Device name (DT, PR, PP, MT, CR, LP, KB)
25+	Unit number
26+	Unit number flag
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR value
For the assign call	
NOT A VALID DEVICE The device name specified in bytes 23 and 24 is a logical device name which is currently not assigned.	6
DEVICE NOT AVAILABLE The device specified is currently assigned or in use by another job.	8
For the reassign call	
ACCOUNT OR DEVICE IN USE The device specified is currently open or has an open file.	3
NOT A VALID DEVICE The device name specified in bytes 23 and 24 is a logical device name which is currently not assigned.	6
DEVICE NOT AVAILABLE The device specified is currently assigned to another job or is in use by another job.	8

Example:

```
10 A$ = SYS(CHR$(6%)+CHR$(10%)+SPACE$(4%)+
        CHR$(0%)+CHR$(0%)+SPACE$(14%)+
        "LP" + CHR$(1%)+CHR$(255%))
        ! ASSIGN LP1:  TO CURRENT JOB.
20 X%=4%
30 A$=SYS(CHR$(6%)+CHR$(10%)+SPACE$(4%)+
        CHR$(X%)+CHR$(0%)+SPACE$(14%)+
        "LP"+CHR$(1%)+CHR$(255%))
        ! REASSIGN LP1:  TO JOB # X%.
```

7.2.4.4 Deassign A Device - Not Privileged (F0=11)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(11%), the deassign a device code
3-6	Not used
7-8	Must be 0
9-22	Not used
23-24+	Device name
25+	Unit number
26+	Unit number flag
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
NOT A VALID DEVICE The device name specified in bytes 23 and 24 is not a valid device name.	6

Discussion:

This call performs the same action as the DEASSIGN system command described in the RSTS-11 System User's Guide.

Example:

The following statement deassigns line printer unit 1 which is assigned to the current job.

```
10  A$ = SYS(CHR$(6%) + CHR$(11%) + SPACE$(4%) +
       CHR$(0%) + CHR$(0%) + SPACE$(14%) +
       "LP" + CHR$(1%) + CHR$(255%))
       ! DEASSIGN LP1:
```

7.2.4.5 Deassign All Devices - Not Privileged (F0=12)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(12%), the deassign all devices code
3-30	Not used

Data Returned: No errors are returned.

Possible Errors: No errors are returned.

Example:

The following statement deassigns all devices currently assigned to the job.

```
10  A$ = SYS(CHR$(6%) + CHR$(12%))
```

7.2.4.6 Zero A Device - Both Privileged and Not Privileged (F0=13)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(13%), the zero a device code
3-4	Not used
5-6	Project-programmer number (see note 1)
7-10	Volume ID for volume label (ANSI format MT only)
11-22	Not used
23-24+	Device designator (Disk, magtape, or DECtape)
25+	Unit number
26+	Unit number flag
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
INVALID FILENAME The device specified is a magtape set to ANSI format, and the volume ID specified in bytes 7-10 is either missing or invalid.	2
CAN'T FIND FILE OR ACCOUNT The device specified is disk and the account specified in bytes 5 and 6 does not exist on the device	5
NOT A VALID DEVICE The device name specified is a logical name which is not currently assigned	6
DEVICE NOT AVAILABLE The device is currently assigned or in use by another job or has a file open	8
DEVICE NOT FILE STRUCTURED The device specified does not allow access by file name.	30

Note 1:

Only privileged users can specify an account other than their own account to be zeroed. Any values a non-privileged user specifies in bytes 5 and 6 are forced to the caller's own project-programmer number. Zeroes in bytes 5 and 6 indicate the project-programmer number of the calling program.

Note 2:

When the zero a device SYS call is specified on magtape or DECtape, the entire medium is zeroed without regard to any project-programmer number. On DECtape, the directory is cleared. On magtape, the tape is rewound to LOAD POINT, three end of file marks are written, and the tape is rewound (See Section A.3.).

Example:

```
10  A$=SYS(CHR$(6%)+CHR$(13%)+SPACE$(2%)+
      CVT%$(0%)+SPACE$(16%)+"SY"+CVT%$(0%))
      ! ZERO MY OWN ACCOUNT ON THE SYSTEM.
20  P0%=10% : P2%=20%   !WANT TO ZERO [10,20]
30  A$=SYS(CHR$(6%)+CHR$(13%)+SPACE$(2%)+
      CHR$(P1%)+CHR$(P0%)+SPACE$(16%)+"SY"+
      CVT%$(0%))
      ! ZERO [10,20] ON THE SYSTEM.
      ! IF PROGRAM IS NON-PRIVILEGED, ZEROES
      ! CURRENT ACCOUNT
40  A$=SYS(CHR$(6%)+CHR$(13%)+SPACE$(20%)+
      "MT"+CVT%$(0%))
      ! ZERO MT:
```

7.2.4.7 CTRL/C Trap Enable - Not Privileged (F0=-7)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-7%), the CTRL/C trap enable code
3-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors: No errors are possible.

Discussion:

After this FIP function is executed in the user program, the Run Time System treats the first CTRL/C subsequently typed on any terminal belonging to the job as a trappable error (ERR=28). Upon execution of the trap, control is immediately passed to the numbered program statement which has been designated as the error-handling routine by the last execution of an ON ERROR GOTO statement. After the trap, CTRL/C trapping is disabled. If it is desired that CTRL/C trapping remain in effect, the SYS call must be executed again.

Such trapping of CTRL/C, however, guarantees only that a defined set of statements is executed when CTRL/C is typed. It is not possible to resume execution at the exact point where the CTRL/C occurred.

If certain critical sections of BASIC-PLUS code are to be completely immune to possible CTRL/C aborts, three actions must occur.

a. The job must detach itself from its terminal. See the description of FIP code +7.

b. The program must have CLOSED all channels on which other terminals in the job had been OPENed.

c. The job must have DEASSIGNed any terminal which had been previously ASSIGNed to it. See the description of FIP code +11.

If the three actions occur, program execution under the job proceeds immune to any CTRL/C.

After the job has completed its critical processing in the detached state, one of three actions can occur.

a. The job can kill itself by means of FIP code +8.

b. The job can find a free terminal (presumably the one from which it detached itself) and "force" into that terminal input buffer the character strings needed for logging into the system and attaching the job to the terminal. (See the descriptions of FIP codes -4, +4, and +6.)

c. The job can find a free terminal and use the REATTACH SYS call to attach itself to the terminal. (See the description of FIP code +6.)

The following sample program shows the procedure.

```
10 ON ERROR GOTO 100                           ! GET SET TO TRAP
20 A$=SYS(CHR$(6%)+CHR$(-7%))                  ! ENABLE ^C TRAPPING
30 PRINT "HI ";
40 SLEEP 10%                                   ! WAIT AWHILE
50 GOTO 30

100 IF ERR<>28% THEN ON ERROR GOTO 0
        ELSE RESUME 110                        ! LOOK FOR A CTRL/C
110 PRINT "^C TRAPPED"
120 SLEEP 10%
130 GOTO 20                                    ! GO BACK TO LOOP
32767 END
```

The program prints "HI" at the keyboard every ten seconds until a CTRL/C is typed. Then it prints the "^C TRAPPED" message and a sleep operation for ten seconds before reenabling the CTRL/C trap and printing "HI". The SLEEP statement before reenabling the trap is included to allow the user to type a second CTRL/C and actually stop the program.

Ordinarily, two CTRL/C characters typed very quickly at a terminal stop a program even if CTRL/C trapping is enabled. However, on a lightly loaded system, it is sometimes possible for the program to react quickly enough to the first CTRL/C that the second one can also be trapped. In this situation, the only means of stopping the job is through the kill job SYS call (or the KILL command in the UTILTY program). Thus, after the original trap, the user can stop the program by typing CTRL/C within ten seconds. It is recommended that programs which trap CTRL/C characters be designed to include a certain amount of time after a trap in which a second CTRL/C actually stops the program.

When a CTRL/C is input from a terminal, further output is inhibited, similar to the effect of the CTRL/O. This is true whether the error condition caused by CTRL/C is processed directly by the BASIC-PLUS editor or is handled by the user's program itself. When the CTRL/C error condition is processed by the editor, it reenables output just prior to printing READY. When the CTRL/C error condition is trapped into the user's own error handling routine, the output to the terminal is reenabled just before executing the ON ERROR GOTO statement.

7.2.5 Privileged Utility SYS Calls

The FIP calls described in this section are privileged calls; that is, they can be called only by a privileged user or by a privileged program. (See Section 1.2 for a discussion of privilege.) Any attempts to execute these calls by non-privileged users or programs result in the error ILLEGAL SYS( ) USAGE (ERR = 18). Other errors are specified in the individual descriptons. The functions described in Sections 7.2.5.2 through 7.2.5.11 are used by the UTILTY system program. Examples of their usage can be found in the source code of that program.

7.2.5.1 Special Shutup Logout - Privileged (F0=-16)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-16%), the special shutup logout code
3-30	Not used

Data Returned: No meaningful data is returned

Errors Returned: Refer to the discussion.

Discussion:

This system function logs the current job off the system (as does the FIP system function call code 5) but, in addition, brings the CPU to an orderly halt at address 54 after the job is logged off the system.

Before this FIP call can execute properly, several system conditions must be true. First, one and only one job can be running on the system when the SYS call is invoked. Next, the number of logins allowed on the system must be 1 (that is, LOGINS DISABLED. See Section 7.2.5.6). Next, no disks except the system disk can be mounted. Finally, no files can be open on the system disk.

If all of these conditions are fulfilled, the system shuts down. If any are not true, any attempt to invoke this SYS call results in the error ILLEGAL SYS( ) USAGE (ERR = 18).

This SYS call is used by the SHUTUP program.

7.2.5.2 Date And Time Changer - Privileged (F0=-14)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-14%), the date and time changer code
3	CHR$(D%) where D% is in the required format to generate the date by the function DATE$(D%). See Section 8.8 of the <u>BASIC-PLUS Language Manual</u> for a description of the DATE$ function.
4	CHR$(SWAP%(D%)) where D% is the same value used in byte 3. This generates the high byte of the value used by the DATE$(0%) function.
5	CHR$(T%) where T% is in the required format to generate the time by the function TIME$(T%). See Section 8.8 of the <u>BASIC-PLUS Language Manual</u> for a description of the TIME$ function.
6	CHR$(SWAP%(T%)) where T% is the same value used in byte 5. This generates the high byte of the value used by the TIME$(0%) function.
7-30	Not used.

Data Returned: No meaningful data is returned.

Discussion:

This function changes the monitor date and time of day values which are returned by the DATE$(0%) and TIME$(0%) functions in BASIC-PLUS.

7.2.5.3 Hang Up A Dataset - Privileged (F0=-9)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-9%), the hang up a dataset code
3	CHR$(N%) where N% is the keyboard number of the line to hang up
4	CHR$(S%) where S% is the number of seconds to wait before hanging up the line
5-30	Not used.

Data Returned: No meaningful data is returned.

7.2.5.4 Broadcast To A Terminal - Privileged (F0=-5)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-5%), the broadcast to a terminal code
3	CHR$(N%) where N% is the keyboard number of the terminal to receive the message
4-?	M$ is the message to broadcast; LEN(M$) can be greater than 27. The string must not be null.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE Generated if LEN(M$) is 0.	18

Discussion:

The data broadcast is printed on the destination keyboard. The received message affects any output formatting being performed on the destination keyboard. System programs which use this function are TALK, PLEASE, BACKUP, and QUEMAN and UTILTY.

7.2.5.5 Force Input To A Terminal - Privileged (F0=-4)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-4%), the force input to a terminal code
3	CHR$(N%) where N% is the keyboard number of the terminal to receive the forced input
4-?	I$ is the input string to force to the terminal. The string must not be null. LEN(I$) can be greater than 27.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE Generated if LEN(I$) is 0.	18

Discussion:

The data forced is seen as input by the system. Other system programs besides UTILTY which use this function are BUILD, INIT, and CONTRL.

7.2.5.6 Disable Further Logins - Privileged (F0=-2)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-2%), the disable further logins code
3-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE A non-privileged user attempts to execute this call.	18

Discussion:

This call sets the number of logins allowed on the system to 1. If no jobs are active on the system, one user can successfully log into the system. However, once one user is logged in, any delimiter typed at a logged out terminal returns the NO LOGINS message. This call is used by the UTILTY and SHUTUP programs.

7.2.5.7 Enable Further Logins - Privileged (F0=-1)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-1%), the enable further logins code
3-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE A non-privileged user attempts to execute this call.	18

Discussion:

This call sets the number of logins allowed to the number specified at start-up time - JOBMAX. The enable logins call is used by the UTILTY and INIT programs.

7.2.5.8 Disk Pack And Terminal Status - Privileged (F0=3)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(3%), the disk pack and terminal status code
3	CHR$(N%); the following values of N% determine the resultant action.

Value	Action
Any Odd	Set terminal status. See FIP call 16.
0	Mount a disk pack or cartridge
2	Dismount a disk pack or cartridge
4	Lock out a disk pack or cartridge
6	Unlock a disk pack or cartridge

For Mount, Dismount, Lock, and Unlock

23-24+	Device name
25+	Unit number
26+	Must be 255.

For Mount

7-10+	Pack identification label in Radix-50 format
?	All bytes not specified are ignored.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ACCOUNT OR DEVICE IN USE An attempt to dismount a disk which has an open file	3
NOT A VALID DEVICE Device name specified is not valid.	6
DEVICE HUNG OR WRITE LOCKED An attempt to mount a disk which is not write enabled	14
ILLEGAL SYS() USAGE An attempt by a non-privileged user to execute this call; or an attempt to mount a disk which is aready mounted or which resides in a non-dismounted drive; or disk specified is the system disk.	18
PACK IDS DON'T MATCH An attempt to mount a disk with an incorrect pack label.	20
DISK PACK IS NOT MOUNTED An attempt is made to lock, unlock or dismount a disk which is not mounted.	21
DISK PACK NEEDS 'CLEANING' The storage allocation table on the disk needs to be restructured since the disk was not properly dismounted when it was last used. Disk is logically mounted but cannot be accessed until cleaned by the CLEAN command of UTILTY system program or by the FIP call with code 2.	25
FATAL DISK PACK MOUNT ERROR The disk is beyond recovery. For example, the cluster size is larger than 16 or the storage allocation table is unreadable.	26
DEVICE NOT FILE STRUCTURED An attempt to lock, unlock, or dismount a disk currently opened in non-file structured mode.	30

Discussion:

Note that if byte 3 contains <u>any</u> odd value, the call is interpreted as a set terminal characteristics call and is exactly equivalent to FIP call 16 discussed in Section 7.2.8. This call is used by the SHUTUP, UMOUNT, and BATCH system programs. (The terminal characteristics form of the call is used by the TTYSET program.) For a discussion of disk management on RSTS/E, see Section 7.1.2 of the RSTS/E System Manager's Guide.

7.2.5.9 Clean Up A Disk Pack - Privileged (F0=2)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(2%), the clean up a disk pack code
3-22	Not used
23-24+	Device name A zero in both bytes means the system disk
25+	Unit number
26+	Unit number flag
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt to use this call by a non-privileged user; the device specified is not a disk; the disk is not locked; a file is open on the disk.	18
DISK PACK IS NOT MOUNTED The disk is not yet mounted.	21
CORRUPTED FILE STRUCTURE The link words in the directories are destroyed or completely meaningless.	29

Discussion:

A clean operation on an RK disk cartridge takes up to 30 seconds and on an RP03 disk pack takes up to five minutes. See Section 7.1.2 of the RSTS/E System Manager's Guide, for a discussion of disk management and the clean operation.

7.2.5.10 Change Password/Quota - Privileged (F0=8)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(8%), the change password/quota, kill job, and disable terminal code
3-6	Not used
7-8	Project-programmer number. Zero for both values means the current account. See Section 7.2.3 for an explanation of the value of each byte.
9-12	New password in Radix-50 format. All zeroes mean no change. See Section 7.2.4.1 for a description of converting strings to Radix-50 format.
13-14	CHR$(N%)+CHR$(SWAP%(N%)), where N% is the number of blocks for the quota. Zero in this word means unlimited quota if byte 21 is 255. Otherwise, zero means no change.
15-20	Not used
21	CHR$(255%) if the quota of 0 specified in bytes 13 and 14 is valid rather than no change.
23-24+	Device name
25+	Unit number
26+	Unit number flag
27	Not used
28	Must be CHR$(0%)
29-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
CAN'T FIND FILE OR ACCOUNT The account is not present on the disk specified.	5
NOT A VALID DEVICE Device specified is not valid.	6
ILLEGAL SYS() USAGE An attempt by a non-privileged user to execute this call or the device specified is not a disk.	18

Discussion:

Either the password or the quota can be changed individually. Also both can be changed in the same call.

7.2.5.11 Kill Job - Privileged (F0=8)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(8%), the change password/quota, kill job, and disable terminal code
3	CHR$(N%) where N% is the number of the job to kill
4-26	Not used
27	Must be CHR$(0%); this byte differentiates the kill job call from the disable terminal call
28	Must be CHR$(255%)
29-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE The job number specified is 0 or is greater than the system JOB MAXIMUM value. An attempt by a non-privileged user to execute this call.	18

Discussion:

There is only one proper way for a job to terminate itself under programmed control. The job must execute the kill FIP call on its own job number. The kill does all of the clean-up that the logout FIP call (F0=5) does, but this function can be executed under program control by any (privileged) program, whereas the logout call requires certain special conditions. Examples of this SYS call can be found in the ERRCPY, BATCH, SPOOL, and QUEMAN programs.

7.2.5.12 Disable Terminal - Privileged (F0=8)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(8%), the change password/quota, kill job, and disable terminal code
3	CHR$(N%), where N% is the keyboard number of the terminal to disable
4-26	Not used
27	Must be CHR$(255%) to differentiate this call from the kill job call
28	Must be CHR$(255%)
29-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE Keyboard number is greater than the number of terminals on the system (NULINE); Keyboard number corresponds to a line used by a pseudo keyboard; Keyboard number relates to a terminal on a DH11 multiplexer line; The terminal is currently opened or assigned by a job; or The keyboard is the system console terminal (KB0:).	18

Discussion:

This FIP call disables a keyboard line. After this function has been executed, no input from the disabled keyboard is processed or echoed by the system, and output generated for the terminal is ignored. There is no complementary function in RSTS/E. Once a keyboard is disabled, it remains disabled until the next time the system starts time sharing.

Note that this function only disables keyboards connected to the PDP-11 with single line interfaces (KL11, DL11, etc.).

No system program (including UTILTY) currently uses this call. It is included at this point because of its relationship to the change password and kill job FIP calls.

7.2.6 Job Scheduling SYS Calls To FIP

7.2.6.1 Priority, Run Burst And Size Maximum Changer - Privileged (F0=-13)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-13), the priority, run burst, and size maximum changer
3	CHR$(J%), where J% is the job number affected or is -1 to denote the current running job
4	CHR$(A%) where A% is 0% to indicate no change to the parameter in byte 5 or is non-zero to indicate a change to the parameter as specified in byte 5.
5	CHR$(P%) where P% is the value of the running priority and ranges from -128 to +128 in steps of 8.
6	CHR$(A%) where A% is 0% to indicate no change to the parameter in byte 7 or is non-zero to indicate a change to the parameter as specified in byte 7.
7	CHR$(R%) where R% is the run burst.
8	CHR$(A%) where A% is 0% to indicate no change to the parameter in byte 9 or is non-zero to indicate a change to the parameter as specified in byte 9.
9	CHR$(S%) where S% is the maximum size, in 1024-word units, to which a job can expand and is between 2 and the current value for SWAP MAX on the system.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged user to execute this call.	18

Discussion:

This system function allows a privileged user to give a running job an increased or decreased chance of gaining run time in relation to other running jobs, and to determine how much CPU time the job can have if it is compute bound. The CPU time is termed the job's run burst and is measured by the number of clock interrupts during which the job can run if it is compute bound.

The size of a job at log in time is set at 2K and can grow during processing to a size limited by the value of SWAP MAX. SWAP MAX is determined at the start of time sharing operations by the system manager. (Refer to the description of SWAP MAX given in the START and DEFAULT option discussed in Chapter 3.) The maximum size to which a job can grow can never be greater than the currently assigned value of SWAP MAX, which must be between 8K and 16K words. Therefore, the privileged user has the option of limiting the size to which a job can grow by specifying a value for S% between 2 and the maximum of SWAP MAX.

Values for each of the variables in the parameter string must be specified. The value for A preceding the related parameter variable determines whether that parameter changes or remains unchanged.

The PRIOR system program provides a direct example of the use of this FIP call. The call is also used by the LOGIN, SPOOL, and RJ2780 system programs.

No error-checking is done by the system on the data passed by the user. Values are used as passed even if they generate illogical results. For instance, if a priority is specified which is not a multiple of 8, its value is truncated to the next lowest multiple of 8. A priority greater than 128 is considered negative. Setting a job's run burst to 0 prevents the job from obtaining any run-time. Setting a (compute-bound) job's run-burst to some high number tends to lock out other jobs. However, setting S% to 255, or any value greater than the system SWAP MAX does not override the system maximum.

7.2.6.2 Set Special Run Priority - Privileged (F0=-22)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-22%), the set special run priority call
3-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged user to execute this call.	18

Discussion:

This SYS function sets the special run priority bit in the job priority word. This action raises the priority of the job slightly above that of other jobs in its priority class. The priority bit is cleared whenever the job returns to the READY state. Thus, a privileged job can raise its priority without protecting against a user typing CTRL/C and retaining the higher priority. This FIP call is used by the QUE system program.

7.2.6.3 Lock/Unlock Job In Memory - Privileged (F0=-20)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-20%), the lock/unlock job in memory code
3	CHR$(N%) where N% is 0 for lock and is 255 for unlock
4-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged user to execute this call.	18

Discussion:

This call prevents unnecessary swapping by forcing the job executing the call to remain in memory. This action is performed without affecting the job priority or run burst. The call merely eliminates the swapping time between run bursts.

A program having certain time sensitive routines can lock itself in memory. The duration of the locked time must be very short to prevent degradation of system performance. Depending on the memory configuration, a locked job can prohibit the system from swapping any other job into memory. If the job expands its size in memory, the system can swap it out of memory regardless of its locked status.

The following sample code demonstrates the lock and unlock procedure.

```
10  A$ = SYS(CHR$(6%) + CHR$(-20%) + CHR$(0%))
    ! LOCK JOB IN MEMORY

100 A$ = SYS(CHR$(6%) + CHR$(-20%) + CHR$(255%))
    ! UNLOCK JOB FROM MEMORY
```

7.2.6.4 Drop Temporary Privileges - Privileged (F0=-21)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-21%), the drop temporary privileges code
3-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged user to execute this call.	18

Discussion:

This call allows a compiled program which has a privileged protection code to drop its temporary privileges. A program normally executes this call after it has used the special privileges to set itself up. The program can take advantage of built-in monitor protections (for example, file protection code arbitration) which are otherwise overridden by a program's temporary privileges. The call does not affect the permanent privileges of an account. Both the QUE and PIP system programs use this call.

The following sample code shows how a program might drop its temporary privileges.

```
10  A$ = SYS(CHR$(6%) + CHR$(-22%)
    ! SET SPECIAL RUN PRIORITY

20  OPEN "SYSTEM.FIL$" AS FILE 1%
    ! OPEN REFERENCE FILE, REGARDLESS OF PROTECTION

30  A$ = SYS(CHR$(6%) + CHR$(-21%)
    ! DROP TEMPORARY PRIVILEGES

40  OPEN "ACCT.FIL" AS FILE 2%
    ! THIS FAILS IF FILE IS PROTECTED AGAINST THE
    ! CURRENT ACCOUNT
```

7.2.7 Account Creation And Deletion SYS Functions

7.2.7.1 Create User Account - Privileged (F0=0)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(0%), the create user account code
3-6	Not used
7-8	Project-programmer number; see Section 7.2.3. The project number can be between 1 and 254; the programmer number can be between 0 and 254.
9-12	Password in Radix-50 format; see Section 7.2.4.1 for a description of converting a string to Radix-50 format.
13-14	Disk Quota. See Section 7.2.3 for a description of unsigned numbers. Zero means unlimited quota.
15-22	Not used
23-24+	Device name
25+	Unit number
26+	Unit number flag
27-28	User file directory (ufd) cluster size; 0 means use the pack cluster size.
29-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL FILE NAME Password is missing in the call	2
NO ROOM FOR USER ON DEVICE The directory currently has the maximum number of accounts.	4
PROTECTION VIOLATION The project-programmer number is [0,0] or either the project or programmer number is 255.	10

NAME OR ACCOUNT NOW EXISTS The account specified in the call currently exists on the device specified.	16
ILLEGAL CLUSTER SIZE The cluster size specified in the call is either greater than 16 or is non-zero and less than the pack cluster size. See Section 5.4.3 for a discussion of valid cluster size values.	23
DEVICE NOT FILE STRUCTURED The device specified is not a disk or the disk is open in non-file structured mode.	30

Discussion:

This call is used by the REACT system program to create accounts.

7.2.7.2 Delete User Account - Privileged (F0=1)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(1%), the delete user account code
3-6	Not used
7-8	Project-programmer number. This call generates an error if account [0,0], [0,1], or [1,1] is specified. See Section 7.2.3 for an explanation of the value of each byte.
9-22	Not used
23-24+	Device name; must be a disk
25+	Unit number
26+	Unit number flag
27-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ACCOUNT OR DEVICE IN USE The account contains files (it has not been zeroed) or a user is currently logged into the system under the account.	3
CAN'T FIND FILE OR ACCOUNT The account specified does not exist.	5
PROTECTION VIOLATION Account specified is either [0,0], [0,1], or [1,1].	10
DEVICE NOT FILE STRUCTURED Device specified is not a disk or is a disk open in non-file structured mode.	30

Discussion:

The REACT system program uses this call to delete user accounts. To prevent error number 3, the user must first zero the account using either the /ZE option in the PIP system program or the ZERO command of the UTILTY system program. The FIP call (F0=13) can also zero an account.

7.2.8 Set Terminal Characteristics Privileged (F0=16)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(16%), the set terminal characteristics code; if CHR$(3%), byte 3 must be odd.
3	If byte 2 is CHR$(3%), this byte must be CHR$(N%) where N% is an odd value.
4	CHR$(N%) where N% is 255% for the current keyboard or is the keyboard number of the terminal to alter.
5	CHR$(N%) where N% is 0% for no change or is the terminal width plus 1. The call sets the number of characters per line to N-1 where N% can be between 2 and 255. The WIDTH n command sets this byte.
6	CHR$(N%) where N% is: 0 for no change 128 to enable hardware horizontal tab feature. The TAB command sets this characteristic. (Device must have the requisite hardware.) 255 to enable software horizontal tab positions which are set every 8 character positions, beginning at position 1. The NO TAB command sets this characteristic.
7	CHR$(N%) where N% is: 0 for no change 128 to enable the software to perform form feed and vertical tab operations by executing four line feed operations. The NO FORM command sets this characteristic. 255 to enable hardware form feed and vertical tab. The FORM command sets this characteristic.
8	CHR$(N%) where N% is: 0 for no change 128 to allow terminal to receive and print lower case alphabetic characters. The LC OUTPUT command sets this characteristic. 255 to have the system translate lower case alphabetic characters to upper case before transmitting to a terminal. The NO LC OUTPUT sets this characteristic.

9 CHR$(N%) where N% is:

- 0 for no change
- 128 to have terminal not respond to XON CHR$(17) and XOFF CHR$(19) characters because it lacks the requisite hardware. The NO XON command sets this characteristic.
- 255 Terminal has requisite hardware to respond to XON and XOFF characters. Terminal stops sending characters when it receives a CHR$(19) character (XOFF) and resumes sending characters when it receives a CHR$(17) character (XON). The XON command sets this characteristic.

10 CHR$(N%) where N% is:

- 0 for no change
- 128 to have characters typed at the terminal sent to the computer only. The computer echoes (transmits back to the terminal) what it receives and performs any necessary translation. The FULL DUPLEX command sets this characteristic.
- 255 to have the terminal (or its acoustic coupler) locally echo the characters typed. The computer does not echo the characters received. The LOCAL ECHO command sets this characteristic.

11 CHR$(N%) where N% is:

- 0 for no change
- 128 Terminal does not have features of a video display terminal. The NO SCOPE command sets this characteristic.
- 255 Terminal is a video display, or cathode ray tube (CRT) type, and uses the following features:

  a. Responds to synchronization standard described by byte 17.

  b. The system executes a DEL character (RUBOUT) by sending a backspace, a space, and a backspace to the terminal.

  c. Any location on the screen can be addressed by direct cursor placement.

  The SCOPE command sets this characteristic.

12 CHR$(N%) where N% is:

0 for no change

128 System treats certain characters received as follows:

a. Translate CHR$(125) and CHR$(126) into the ESC character CHR$(27).

b. Translate lower case characters to upper case characters.

Set by NO LC INPUT command.

255 Terminal transmits the full ASCII character set and system treats special characters as follows:

a. Treat only CHR$(27) as an escape character (echoed as the $ character and handled as a line terminating character).

b. Treat CHR$(125) and CHR$(126) as printed characters and

c. Do not translate lower case characters to upper case format.

Set by LC INPUT command.

13 CHR$(N%) where N% is:

0 for no change

n Set fill factor of terminal to N%-1. The command FILL n determines this value.

255 Set fill factor for an LA30S (serial) DECwriter. The command FILL LA30S sets this characteristic.

14 CHR$(N%) where N% is:

0 for no change

n The internal speed value to determine the baud rate at which the terminal receives characters. If byte 16 is 0, this value also determines the transmit (output) baud rate. If byte 16 is 255, this byte must be 255. For a DH11 line, n is between 1 and 16. For a DC11 line, n is between 1 and 4. See the <u>PDP-11 Peripherals Handbook</u> for the related baud rates.

255 2741-type terminal. Byte 16 must also be 255%.

15 CHR$(N%) where N% is:

0 for no change

1 Do not set the output parity bit. This value is set by the NO PARITY command.

2 Set the output parity bit for even parity format. The EVEN PARITY command sets this value.

3 Generate an output parity bit for odd parity format. The ODD PARITY command sets this value.

16 CHR$(N%) where N% is:

- 0 Both the receive (input) and transmit (output) speeds are determined by n in byte 14.
- n The internal speed value to determine the baud rate at which the terminal transmits characters when a split speed setting is used. For a DH11 line, n is between 1 and 16; for a DC11 line, n is between 1 and 4.
- 255 2741-type terminal. (See description of byte 20.) Byte 14 must also be 255%.

17 CHR$(N%) where N% is:

- 0 for no change
- 128 Terminal ignores synchronization standards described for 255% value. The NO STALL command sets this characteristic.
- 255 Terminal obeys the synchronization standard as follows. The computer stops sending characters if the terminal transmits a CHR$(19) character (XOFF, or the CTRL/S combination). Computer resumes sending characters when the terminal transmits a CHR$(17) character (XON, or the CTRL/Q combination). The STALL command sets this characteristic.

18 CHR$(N%) where N% is:

- 0 for no change
- 128 System echo prints a control character received as the up arrow (^ or) character followed by the equivalent printable character. For example, the CTRL/D combination is printed as ^D, CHR$(94) followed by CHR$(68). The UPARROW command sets this characteristic.
- 255 System treats control characters as such. The NO UPARROW command sets this characteristic.

19 CHR$(N%) where N% is a value between 1 and 30 and determines the maximum number of characters in a burst for a DH11 line. For low speed lines, this is set to 30; for VT50 lines, it is set to 12. The DH BURST n command sets this value.

20 CHR$(N%) where N% depends on the values of two other bytes. If bytes 14 and 16 are both 255, the value of this byte applies to the 2741-type terminal as follows:

```
n = 8 + DATA+STOP+PARITY
```

where:

```
   DATA is 0 for 5 bits per character
           1 for 6 bits per character
           2 for 7 bits per character
           3 for 8 bits per character

   STOP is 0 for 1 stop bit per character
           4 for 2 stop bits per character
            or 1.5 bits if DATA=0.

   PARITY is 0 for no parity bit.
             16 for even parity format
             48 for odd parity format.
```

If either byte 14 or 16 is other than 255, this byte is not used. The 2741 command determines the value of this byte.

21 CHR$(N%) where N% is:

0	for no change
255	Set the ring list entry for a terminal attached to a DC11, DL11E or DH11 line interface to default to permanent characteristics when modem is answered. The /RING option with a TTYSET KBn: command determines this value.

22 CHR$(N%) where N% is:

0	for no change
128	The system software treats an incoming ESC CHR$(27%) character as a line terminating character and echoes it as the $ character. The NO ESC SEQ command sets this value.
255	The software treats an incoming ESC CHR$(27%) character and the next incoming character as a special escape sequence. The software does not echo either character but transposes the characters and changes the ESC character to a CHR$(155%) character. That is, if the incoming data is:

```
ESC      X
27      XXX
```

then the user receives:

```
 X      ESCAPE
XXX     155
```

The ESC SEQ command sets this value.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE The keyboard number specified in byte 4 of the call is out of range of the valid keyboard numbers.	18

Discussion:

If the terminal specified by the keyboard number in byte 4 of the call either is disabled (as a result of the system initialization procedure or of executing the disable terminal SYS call) or is a pseudo keyboard, the call is not executed by the system.

The TTYSET system program employs this call to set terminal characteristics. Refer to the discussion of TTYSET in Section 4.5 of the RSTS-11 System User's Guide.

7.2.9 Change File Statistics

Privileged F0=-11)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-11%), change file statistics code
3	CHR$(N%) where N% is the internal channel on which the file is open. Must be between 1 and 12, inclusive
4-5	Desired date of last access
6-7	Desired date of creation
8-9	Desired time of creation
10-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE The file open on the channel specified is not a disk file or is a user file directory. An attempt by a non-privileged user to execute this call.	18

Discussion:

The data passed by this call replaces the related data in the accounting block of the file open on the channel specified in byte 3. No error checking is done on the date and time values passed. Since the call does not supply default values, the user program must supply all three date and time values each time the call executes. The call is used by the BACKDK system program to maintain original date and time statistics for a file.

The following is a partial directory listing of a privileged account showing the file whose statistic information is to be changed.

```
CAT
CTPBLD.BAS        2        60      04-SEP-74 03-MAY-74 09:00 PM
```

The following program changes the date of creation to 22-July-74, and the date and time of last access to 12:00 noon, 22-July-74, as shown on the partial directory listing following the program.

```
LISTNH
10      OPEN "CTPBLD.BAS" AS FILE 1%       ! OPEN FILE TO CHANGE.
20      DIM M%(30%)                        ! USE THIS ARRAY TO SET UP CALL
30      M%(0%)=30%
40      M%(1%)=6%
50      M%(2%)=-11%                        ! SET UP FOR CHANGE STATS CALL
60      M%(3%)=1%                          ! ON CHANNEL 1.
70      M%(4%)=4203% AND 255%
        : M%(5%)=SWAP%(4203%) AND 255%     ! SET UP DATE OF CREATION
                                           !  FOR 22-JUL-74 (DATE$(4203%)=
                                           !  22-JUL-74).
80      M%(6%)=4203% AND 255%
        : M%(7%)=SWAP%(4203%) AND 255%     ! SET UP LAST ACCESS TO 22-JUL-74.
90      M%(8%)=720% AND 255%
        : M%(9%)=SWAP%(720%) AND 255%      ! SET UP TIME OF LAST ACCESS TO 12 NOON
                                           !  (TIME$(720%)=12:00 PM)
100     CHANGE M% TO M$                    ! SET IT UP AS A STRING.
110     M$=SYS(M$)                         ! AND DO THE WORK
32767 END

READY

RUNNH

READY

CAT
CTPBLD.BAS        2        60      22-JUL-74 22-JUL-74 12:00 PM


READY
```

7.2.10 LOGIN And LOGOUT SYS Calls

7.2.10.1 LOGIN - Privileged (F0=4)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(4%), the LOGIN code
3-6	Not used
7-8	Project-programmer number. Must not be account [0,1]. See Section 7.2.3 for an explanation of the value of each byte.
9-12	Password in Radix-50 format. See Section 7.2.4.1 for a description of converting a string to Radix-50 format.
13-30	Not used.

Data Returned:

Byte(s)	Meaning
1-2	Internal data
3	Total number of jobs logged into the system under this account
4-?	Job numbers of each job running detached under this account. A byte of CHR$(0%) signifies the end of the list. Only the first 26 job numbers are returned.

Possible Errors:

Meaning	ERR Value
CAN'T FIND FILE OR ACCOUNT The project-programmer number specified in the call is [0,1], does not exist, or its password does not match the password of the account on the system.	5

Discussion:

If the calling job is already logged into the system, this call does <u>not</u> change the job's account. The data returned in bytes 3 through 30 refers to the same account under which the job is running.

7.2.10.2 LOGOUT - Privileged (F0=5)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(5%), the LOGOUT code
3-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE The JFHIBY value is not set when the call is executed.	18

Discussion:

This call closes all open channels, deassigns all devices, updates statistics on the disk, clears the job from the monitor message table and disassociates the project-programmer number from the job number. However, none of these actions are performed unless the monitor sets a special flag called JFHIBY.

The monitor sets JFHIBY only when a user types the HELLO, BYE or ATTACH commands at a terminal logged into the system or types anything at a terminal not logged into the system. No way exists for a BASIC-PLUS program to set JFHIBY. Thus, an already running program can never log itself off the system.

When the monitor sets the JFHIBY flag, the running program has full privileges. The drop temporary privileges call resets the JFHIBY flag to disable the privileges afforded by the flag. The LOGOUT and LOGIN system programs use the logout call.

7.2.11 Detach, Attach, And Reattach SYS Calls

7.2.11.1 Detach - Privileged (F0=7)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(7%), the detach code
3-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE The current job is already detached or its console keyboard is open on a channel other than channel 0.	18

Discussion:

This call disassociates the job and its console keyboard. The following sample program segment prints a message and detaches from the keyboard.

```
100  PRINT "DETACHING..."
     ! NOTIFY THE USER
110  A$ = SYS(CHR$(6%) + CHR$(7%))
     ! DO THE DETACH
```

When data is entered at a RSTS/E terminal, the system activates a job to handle the input and gives the job the next available job number. If the data is recognized by the system, certain actions are executed under that job number, one of which can be logging a user into the system. (See Sections 2.1 and 4.1 of the RSTS-11 System User's Guide for the operational details.) When a user is logged into the system, the activated job is associated by the system with both the terminal at which he is typing and the account number which he used to identify himself to the system. The job is then considered active on the system and in attached mode, or, simply, attached to the terminal.

A privileged job can become detached from the terminal by executing this call. Once a job is placed in the detached state, it runs as any other job logged into the system but it does not employ a terminal device on channel 0. The detached state is advantageous for non-interactive jobs. The job running detached frees a terminal for other usage and makes the job immune from interruption by someone typing a CTRL/C combination.

When the user desires to attach a detached job to a terminal, he can log into the system at any free terminal using the account number under which the detached job was made active and attach that job to the terminal. (This procedure is described in Section 4.1 of the RSTS-11 System User's Guide.) Since the system associated the job number of the attached job with the account number under which that job was made active, it reports the detached job under the same account number.

This attachment facility is valuable in another manner. A job can be placed in a detached state by the system when the carrier is dropped on a remote line. This means that, if the telephone connection is lost while a job is running from a terminal at a remote location, the content of the job is not lost. The user simply logs into the system again with the same account number and reattaches to the job he was previously running.

The ERRCPY, QUEMAN, SPOOL, and BACKUP system programs use this call.

7.2.11.2 Attach - Privileged (F0=6)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(6%), the attach and reattach code. The code to attach and reattach is the same but the format of the data passed is quite different. See Section 7.2.11.3 for the reattach format.
3	The number of the job to attach to the terminal.
4	Must be 0
5-6	Project-programmer number of the job to attach to the terminal. See Section 7.2.3 for a description of the exact contents of each byte.
7-10	Password of the account specified in bytes 5 and 6 in Radix-50 format. See Section 7.2.4.1 for information on converting a string to Radix-50 format.
11-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE Each of the following conditions generates this error:	18

1. The job executing the call has an open channel.
2. The job executing the call is a source (BAS) program rather than a compiled (BAC) program.
3. The job number specified in byte 3 is not a detached job.
4. The account or password in the call is not valid.
5. The job executing the call is detached.

Discussion:

The LOGIN system program is the only program that can execute this call. See the LOGIN.BAS listing for an example of its usage. Note that, if byte 3 is the number of the job executing the call, the system performs the reattach action. See Section 7.2.11.3 for a description of the reattach process.

7.2.11.3 Reattach - Privileged (F0=6)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(6%), the attach and reattach code. The code to attach and reattach is the same but the format of the data passed is quite different. See Section 7.2.11.2 for the attach format
3	CHR$(J%) where J% is the number of the job executing the call
4	CHR$(K%) where K% is the keyboard number of the terminal to which the calling job is to be attached
5-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE Each of the following conditions generates this error:	18

1. The job number specified in byte 3 is less than 1 or greater than the JOB MAX value on the system.
2. The job executing the call is not detached.
3. The keyboard number in byte 4 is out of range.
4. The terminal specified by the keyboard number in byte 4 is currently assigned, opened, or the console keyboard of some job.
5. The job executing the call has an open file.
6. The job executing the call is a source (BAS) program rather than a compiled (BAC) program.

Discussion:

A privileged job can execute this form of the attach and reattach call. The call establishes the terminal specified in byte 4 as the console keyboard of the detached job executing the call. In this manner, a job can reattach to a terminal without having to force the proper data to the desired terminal.

7.2.12 Send And Receive Messages

Both Privileged (F0=18) and Not Privileged

This function call allows a user's job to do the following:

a. declare itself a receiving job,

b. send a message to a receiving job,

c. receive a message from pending messages,

d. stall until a message is pending, and

e. eliminate itself or another job as a receiving job.

The monitor controls eligibility to receive messages by maintaining a table of receiving jobs. The response to the RECEIVERS query at system generation time determines the maximum number of jobs eligible to receive messages at any given time. To be eligible to receive messages, a job must declare itself and have its identification entered in the table of receiving jobs. A job sending a message succeeds only if the job to which the message is sent has an entry in the table.

Sending and receiving messages on the system uses small buffers. Each message occupies one small buffer. A job defines, in its first receive call, the number of pending messages (messages sent to the job but not yet received by the job) which monitor allows to be queued for the job at any given time. This maximum can be as high as 127. To prevent occupying a large number of small buffers, and thereby degrading system performance, the pending maximum for each receiver should be small (10 to 15) and each receiving job should extract its messages quickly.

The system controls message operations on a first-in, first-out (FIFO) basis. It maintains pending messages as a linked chain of small buffers. When a job sends a message to an eligible receiving job, the system appends the related small buffer to the last small buffer in the chain of messages pending for the job. When a receiving job asks for a pending message, the system makes available the first message in the chain and removes the related small buffer from the chain.

The system continues message operations for a receiving job until either the maximum number of messages are pending or the supply of small buffers is exhausted. Since such conditions affect system operations, a receiving job must process its pending messages frequently to maintain adequate system performance. Since poorly designed use of the receive mechanism can drastically degrade overall system performance, the receive operation can be executed only by a privileged user.

A receiving job must remove itself from the table of receiving jobs or have another job remove it. To keep non-active jobs from occupying entries in the table of receivers, both the logout SYS call and the kill job SYS call remove the job from the table of eligible receiving jobs.

7.2.12.1 Declaring A Receiver And Receiving A Message - Privileged (F0=18)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(18%), the send and receive a message code
3	CHR$(N%) where N% is one of the following values: 1 Attempt to receive a message or declare this job as a receiver and attempt to receive a message. Return an error condition if no messages are pending. 2 Receive with sleep. Similar to 1 except that the job executes a sleep operation if no messages are pending. This action occurs in place of generating an error condition.
4	CHR$(P%+L%) where P% can be either 0 or 128. If P is 0, messages can be received from any sending job. If P is 128, messages can be received from and queued only by sending jobs which are privileged. L% is the number of messages (between 1 and 127) which can be simultaneously pending for this receiving job.
5-8	Receiving job logical name in Radix-50 format. See Section 7.2.4.1 for a description of converting a string to Radix-50 format.
9-30	Not used.

Data Returned:

Byte(s)	Meaning
1-4	Not used
5	CHR$(J%) where J% is the job number times 2 of job sending the message
6	Must be CHR$(0%)
7-8	Project-programmer number of the pending job. See Section 7.2.3 for a description of each byte.
9-28	The message string. The system pads any unused bytes with NUL characters to a length of 20 bytes.
29-30	Not used.

Possible Errors:

Meaning	ERR Value
NO ROOM FOR USER ON DEVICE When the job attempts to declare itself as a receiving job, the monitor table containing data of eligible receiving jobs is full or is zero length.	4
CAN'T FIND FILE OR ACCOUNT For receive only, error indicates no messages are pending. For receive with sleep, error indicates no messages were pending when the receive was executed by monitor. Error is returned to the program when monitor awakens job from sleep.	5
ILLEGAL SYS() USAGE When the job attempts to declare itself as a receiving job, either the logical name is missing from the call (bytes 5 through 8 are not given) or another job has already declared itself a receiver with the logical name given.	18

Discussion:

A receive call checks the eligibility of a job to receive messages and performs both or one of two actions based on the result of the check. First, if the job is not eligible to receive messages, the call declares the job as an eligible receiver and attempts to receive a message. Second, if the job is already eligible, the call attempts to receive a message. Since the same call performs two actions, it is important that the user program handle the declaration and receiving procedure properly.

To check the eligibility of a job to receive messages, the system determines if the calling job's job number appears in a table in the monitor part of memory. If the job number is not in the so called receiver table, the system ensures that the logical name specified in bytes 5 through 8 of the data passed is not currently being used by another job. If the logical name is unique and an empty slot is available, the system declares the job as a receiver by entering in the table its job number, its logical name, and other data. Subsequent to declaring the job as a receiver, the receive call never refers to the logical name. The logical name exists so that a sending job can easily refer to a receiver without supplying its job number.

A receive call can not change a logical name in the table of eligible receivers because the system refers to the job number in the receive table rather than to the logical name. If the job number of the current job is in the table, the system considers the job eligible and has no need to refer to the logical name. This condition is important if a previous receiver with the same job number as the current receiver failed to remove itself from the table before terminating processing. Thus, a logical name already appears in the table for the current receiver when it attempts to declare itself a receiver.

To eliminate the possibility of a spurious logical name appearing in the table for the current job, it is recommended that a program execute the call to remove itself as a receiver before it executes the first receive call to declare itself a receiver. In this manner, the program ensures that the logical name in the table for the current job is, in fact, the name declared in bytes 5 through 8 of the call.

When the receive call declares a job as a receiver, it also attempts to receive a message. Because the system does not queue messages for a job which is not an eligible receiver, the first attempt to receive a message always fails.

When the receive call determines that a job is eligible, it attempts to receive a message. If a message is pending for the job, the call returns the information in bytes 9 through 28 of the target string. If no message is pending for the job, the call executes according to the value of N% in byte 3 of the data passed. If the value of N% is 1, the call immediately generates a recoverable error (ERR = 5). If the value of N% is 2, the call puts the job in a SLEEP state (called a receiver sleep).

The system awakens a job in a receiver sleep if a message becomes queued for it or if a line terminating character is typed on one of its keyboards. Since the system presets the recoverable error condition (ERR = 5) before putting the job to sleep, the receiving job, upon awakening, detects the error condition. The system does not pass the message to the job. To obtain the message queued, the job must execute the receive call again.

7.2.12.2 Send A Message - Both Privileged and Not Privileged (F0=18)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(18%), the send and receive a message code
3	CHR$(-1%); this value indicates that the call is a request to send a message
4	CHR$(J%) where J% is the job number times 2 of the job to receive the message. If J% is 0, the call uses the logical name in bytes 5 through 8 to determine the receiving job
5-8	Receiving job name in Radix-50 format. See Section 7.2.4.1 for a description of converting a string to Radix-50 format.
9-28	Message text to send. Can be a maximum of 20 bytes and the system pads the message with NUL characters to the length of 20 bytes
ɔ-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
CAN'T FIND FILE OR ACCOUNT The receiving job specified is not in the monitor table of eligible receiving jobs.	5
ILLEGAL SYS() USAGE The receiving job specified is capable of receiving messages only from privileged jobs and the sending job is not privileged.	18
NO ROOM AVAILABLE FOR FCB One of two conditions is possible. 1. The number of messages pending for the receiving job is at its declared maximum. The sending job must try again. If this condition occurs frequently, it indicates that the declared maximum is too low or that the receiving job is not processing its messages quickly enough. 2. Also, sending a message requires a small buffer and one is not available.	32

Discussion:

The SEND operation sends a message to a declared receiving job in one of two ways: by either a job number or by a job identification. When byte 4 of the data passed is non-zero, the call ignores bytes 5 through 8 and attempts to send the message to the job designated by the value in byte 4. If byte 4 is zero, the call attempts to send the message to the receiver whose identification matches that given in bytes 5 through 8.

The sending job can be either privileged or non-privileged. The sending job must be privileged if the receiver is capable of receiving messages only from privileged sending jobs. (The receiver determines this capability by specifying a proper value in byte 4 of the Declare a Receiver SYS call.) If a non-privileged sender attempts to send a message to a receiver which is accepting messages only from privileged sending jobs, the monitor does not queue the message and returns the ILLEGAL SYS() USAGE error to the non-privileged sender. The sending job can be either privileged or non-privileged if the receiver is capable of receiving messages from any sending job.

7.2.12.3 Removing A Receiver - Privileged (F0=18)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(18%), the send and receive a message code
3	CHR$(0%) to remove a receiving job from the monitor table of receiving jobs
4	CHR$(N%) where N% is the number of the job to remove or is 0 to remove the job executing the call
5-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged job to execute this call.	18

Discussion:

This function removes the job number and logical name from the receive table. All pending messages are lost.

7.2.13 Poke Core Privileged (F0=-6)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-6%), the poke core code
5-6	CHR$(A%)+CHR$(SWAP%(A%)) where A% is the address to change
	CHR$(V%)+CHR$(SWAP%(V%)) where V% is the value to insert at the address specified by bytes 3 and 4.
7-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
PROTECTION VIOLATION The job executing the call is not operating under account [1,1] or the address specified in the call is an odd value.	10

Discussion:

This call changes a word in the monitor part of memory to the value the user specifies. Obviously, this is a very dangerous capability, and it is, therefore, heavily protected. It can only be called from a job running on account [1,1].

The poke call allows only full word changes. If the user desires a byte change, he must read the word (using the PEEK function), change the desired byte, and rewrite (using the POKE call) the entire word.

7.2.14 Set Logins Privileged (F0=-19)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-19%), the set logins code
3	CHR$(N%) where N% is the number of logged in jobs to allow
4-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged job to execute this call.	18

Discussion:

This function sets the number of allowable logins to the number specified in byte 3. If N is 0, the number set is 1. If N is greater than the system JOBMAX set at start up time, then the number set is the value of JOBMAX.

7.2.15 Accounting Information

7.2.15.1 Read Or Read And Reset Accounting Data - Both Privileged and Not Privileged (F0=14)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(14%), the read or read and reset accounting data code
3-4	CHR$(N%)+CHR$(SWAP%(N%)) where N% is the index number of the account to read. If N% is 0, read the account specified in bytes 7 and 8.
5-6	CHR$(N%) where N% is 0 to indicate read only and is non-zero to indicate read and reset. If the job executing this call is not privileged, the system does not access this word and performs only a read operation.
7-8	Project-programmer number. Used only if bytes 3 and 4 are 0. See Section 7.2.3 for a description of each byte.
9-22	Not used
23-24+	Device name; must be a disk. A zero in both bytes indicates the system disk.
25+	Unit number
26+	Unit number flag
27-30	Not used

Data Returned:

Byte(s)	Meaning
1-4	Internal coding
5-6	Number of blocks owned by the account read
7-8	Project-programmer number of the account read
9-12	Password of the account read; in Radix-50 format
13-14	Low order word (16 bits) of the CPU time (in tenths of seconds) used by the account
15-16	Connect time (in minutes) used by the account
17-18	Low order word (16 bits) of kilo-core ticks used by the account
19-20	Device time (in minutes) used by the account
21-22	High order bits for CPU time and kilo-core ticks. See the discussion for an explanation of how the values are stored.
23-26	Device and unit information unchanged
27-28	Disk quota in number of blocks; 0 means unlimited quota
29-30	User file directory cluster size.

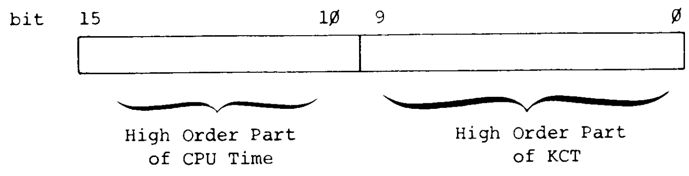
Possible Errors:

Meaning	ERR Value
CAN'T FIND FILE OR ACCOUNT The project-programmer number specified does not exist on the disk or the index specified is greater than the number of accounts on the disk.	5
ILLEGAL SYS() USAGE Device specified is not a disk.	18

Discussion:

This FIP call is the only one provided in RSTS/E to lookup accounts on a disk. By starting the index (bytes 3 and 4) at 1 and incrementing it for each call, the user program can retrieve the project-programmer number of every account on the disk. See the description of the MONEY system program (Section 6.5) for a discussion of the accounting information.

The word returned in bytes 21 and 22 holds the high order bits of CPU time and kilo-core ticks. The bottom ten bits of this word apply to kilo-core ticks, and the top six bits apply to CPU time. Graphically, the word looks like the following:



If a non-privileged program executes this call, the system forces the following bytes in the data passed to the values shown.

3 and 4	0	Look up the account specified in bytes 7 and 8
5 and 6	0	Read only
7 and 8	current PPN	Look up data for current project-programmer number

If a privileged program executes this call and bytes 5 and 6 of the data passed are non-zero, the following account information is read and reset to zero.

CPU time
kilo-core ticks
connect time
device time

7.2.15.2 Accounting Dump - Privileged (F0=-15)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-15%), the accounting dump code
3-4	Project-programmer number of the account to which the system dumps the accumulated usage data. See Section 7.2.3 for a description of each byte. If both bytes are zero, the data is dumped to the current account.
5-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
CAN'T FIND FILE OR ACCOUNT The account specified in bytes 3 and 4 does not exist.	

Discussion:

This function allows a program to dump accumulated accounting data to the account specified in bytes 5 and 6. This capability enables user callable utility programs to run on an account different from the account which called them and still charge the calling account for the time accumulated by the utility. For example, the SPOOL program must run on a privileged account, but is callable by non-privileged users through the QUE command. It is desirable that the calling user be charged for the time the SPOOL program accumulates processing the job. The SPOOL program could take advantage of this SYS call to effect this process.(1)

----

(1)The version of SPOOL released with RSTS/E does not perform this function.

7.2.16 Directory Look Up

The SYS function calls described in this section look up file names under programmed control. Although only two codes are available, four different types of operation are possible. As a result, four descriptions appear in this section.

The four types of operation return the data in the same format as described below.

Data Returned:

Byte(s)	Meaning
1-4	Not used
5-6	Same as data passed. (Project-programmer number)
7-10	File name in Radix-50 format. See Section 7.2.2 for a description of converting a string in Radix-50 format
11-12	Extension in Radix-50 format
13-14	Length in blocks or sectors. (Not used for special magtape look up described in Section 7.2.16.2)
15	Protection code of the file
16	0
17-18	For disk, the date of last access; for tape, the date of creation
19-20	For disk, the date of creation; for tape, not used
21-22	For disk, the time of creation; for magtape, the project-programmer number of the file
23-26	Same as data passed. (Device name, unit number, and flag byte)
27-28	For disk, the file cluster size; for tape, not used
29	Number of entries returned: for disk, 8; for tape, 6. (Not returned if F0 is 17.)
30	Not used.

7.2.16.1 Directory Look Up On Index - Not Privileged (F0=15)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(15%), the directory look up on index code
3-4	CHR$(N%)+CHR$(SWAP%(N%)) where N% is the index of the file to read. If N% is 0, return the data for the first file in the directory. If N% is x, return the data for the x+1 file in the directory. On magtape, N% must be 0 to rewind the tape before reading the first file. See Section 7.2.16.2 for a description of magtape operations. On DECtape, N% must be 0 to read the directory blocks from the tape before reading the first file. Subsequent calls where N% is not zero read the directory from the BUFF.SYS file.
5-6	Project-programmer number of the directory to look up. If both bytes are 0 and the device specified in bytes 23 and 24 is magtape, the call returns information for each file read. If the device specified in bytes 23 and 24 is DECtape, the call does not use these bytes but returns information for each file read. See Section 7.2.3 for a description of these bytes.
7-22	Not used
23-24+	Device name for look up
25+	Unit number
26+	Unit number flag
27-30	Not used.

Data Returned: See introductory material for Section 7.2.16.

Possible Errors:

Meaning	ERR Value
CAN'T FIND FILE OR ACCOUNT The account specified does not exist on the device specified or no more files exist on the account (the index value is greater than the number of files on the account).	5
DEVICE NOT FILE STRUCTURED The device specified in the call is not a file structured device.	30
VARIOUS DEVICE DEPENDENT ERRORS The call also returns device dependent errors such as DEVICE HUNG and DISK PACK NOT MOUNTED.	

Discussion:

The CATALOG system command employs the same routines as this call to print a directory listing. The ordering of the files in the listing is by index value from the lowest to the highest. The user can therefore determine the index value for a certain file by counting its position in a CATALOG listing and subtracting one.

If the device specified is magtape, the monitor, after reading a file label, skips to the end of the file on the tape to determine the number of blocks in the file.

7.2.16.2 Special Magtape Directory Look Up - Not Privileged (F0=15)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(15%), the directory look up on index code
3-4	CHR$(N%)+CHR$(SWAP%(N%)) where N% is the index of the file to read. If N% is 0, return the data for the first file in the directory. If N% is x, return the data for the x+1 file in the directory. On magtape, N% must be 0 to rewind the tape before reading the first file. See Section 7.2.16.2 for a description of magtape operation. On DECtape, N% must be 0 to read the directory blocks from the tape before reading the first file. Subsequent calls where N% is not zero read the directory from the BUFF.SYS file.
5-6	Both bytes are CHR$(255%) to execute the special magtape directory look up
7-22	Not used
23-24	MT
25+	Unit number
26+	Unit number flag
27-30	Not used.

Data Returned: See introductory material for Section 7.2.16.

Possible Errors:

Meaning	ERR Value
CAN'T FIND FILE OR ACCOUNT No more files exist on the tape.	5
DEVICE NOT FILE STRUCTURED The device specified in bytes 23 and 24 is not file structured.	30

Discussion:

The standard directory look up call (described in Section 7.2.16.1) executed on a magtape unit results in the following actions by the monitor:

a. Reads one record from the tape (a label record).

b. Spaces the tape forward to the next end of file record and calculates the number of records in the file.

c. Returns the directory information if the account number of the file matches the one specified in the call or if both bytes in the account specification in the call are zero.

When the monitor executes the action described in statement a, the tape must be positioned immediately before a label record. Otherwise, an error is generated or garbage information is returned.

In an application program which must search a tape for a specific file and read each specific file found, the OPEN FOR INPUT statement necessitates a rewind operation. The OPEN FOR INPUT statement executed on a file structured magtape generally causes the following actions.

a. Reads one record from the tape which must be a label record.

b. If the read operation is successful, then opens the file and returns control to the user program.

c. If the read operation is unsuccessful and this is the first label read, then rewinds the tape and executes the action described at a.

d. If the logical end of tape is detected, returns an error.

e. If the label read does not match, skips to the end of this file and executes the action at a.

The required rewind operations consume time and are clearly unwanted.

To avoid the rewind operations, the application program can execute the special magtape directory look up call and perform certain actions. By specifying bytes 5 and 6 both as CHR$(255%) in the call, the program causes the following actions by the monitor.

a. Reads from the tape a record which must be a label record.

b. Backspaces one record which leaves the tape in a position to read the label record again.

c. Returns the directory information (except for file length) to the program.

To take advantage of this special action, the program can perform the following actions.

a. Determine from the information returned whether the file is the one required.

b. If the file is required, execute the OPEN FOR INPUT statement using the file name and requesting no rewind. The action executes without a rewind because the tape is positioned properly. If the file is not required, space the tape forward to the next end of file record (see Section 12.3.7 of the BASIC-PLUS Language Manual) before executing another call.

c. After processing the required file, execute a CLOSE statement to position the tape at the end of file record and to be ready to execute another call.

The special look up call returns directory information on each file read regardless of its account number. However, the OPEN FOR INPUT statement must specify the correct account number if the account number of the file does not correspond to the current account number.

7.2.16.3 Disk Directory Look Up By File Name - Not Privileged (F0=17)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(17%), the disk directory look up by file name and disk wildcard directory look up code. See Section 7.2.16.4 for a description of the latter call.
3-4	Both bytes must be CHR$(255%)
5-6	Project-programmer number of the file to look up. See Section 7.2.3 for a description of each byte.
7-10	File name in Radix-50 format. See Section 7.2.4.1 for a description of converting a string to Radix-50 format.
11-12	Extension in Radix-50 format
13-22	Not used
23-24+	Device name; must be disk
25+	Unit number
26+	Unit number flag
27-30	Not used.

Data Returned: See introductory material for Section 7.2.16.

Possible Errors:

Meaning	ERR Value
ILLEGAL FILE NAME File name in bytes 7 through 10 is missing.	2
CAN'T FIND FILE OR ACCOUNT The device specified in bytes 23 and 24 is not a disk or the file specified does not exist on the specified disk.	5

Discussion:

This call works only on disk files and returns information for the specified file.

7.2.16.4 Disk Wild Card Directory Look Up - Not Privileged (F0=17)

Data Passed: Same as that described in Section 7.2.16.3 except for the following data.

Byte(s)	Meaning
3-4	CHR$(I%) + CHR$(SWAP%(I%)). If I% is 0, return the data for the first file which matches the wild card specification. If I% is x, return the data for the x + 1 file which matches the wild card specification.
7-10	Radix-50 representation of a wild card file name specification where an * character can replace the file name or a ? character can replace any character in the file name. Used with the extension in bytes 11 and 12 to create a wild card file specification. See Section 7.2.4.1 for a description of converting a string to Radix-50 format.
11-12	Radix-50 representation of a wild card extension specification were an * character can replace the extension or a ? character can replace any character in the extension. Used with the file name in bytes 7 through 10 to create a wild card file specification. See Section 7.2.4.1 for a description of converting a string to Radix-50 format.

Data Returned: See Introductory material for Section 7.2.16.

Possible Errors:

Meaning	ERR Value
ILLEGAL FILE NAME No file name appears in bytes 7 through 10.	2
CAN'T FIND FILE OR ACCOUNT The device specified in bytes 23 and 24 is not a disk or no match exists for the index value given in bytes 3 and 4.	5
DISK PACK IS LOCKED OUT The disk is in the locked state and the account under which the call is executed is not privileged.	22

Discussion:

This call allows a program to supply a wild card specification and to increment an index value to gain directory information for all occurrences of files matching the wild card specification. The following are typical wild card specifications and their meanings.

FILE??.*	All files with FILE as the first four characters in the name and with any extension (including no extension)
*.BAS	All files with BAS extensions
*.BA?	All files with BA as the first two characters in the extension

The program supplies an index of 0 and executes the call. The system returns directory information for the first file which matches the wild card specification. The program can increment the index by 1 and execute the call again to gain directory information for second and subsequent matching occurrences of files. The system returns error number 5 to indicate no more matching occurrences exist in the account. The entire procedure relieves the program of the overhead required to translate each file name in the directory and to compare for a match.

7.2.16.5 General Guidelines For Usage Of Directory Look Up Calls - The following conditions apply to executing one of the directory look up calls described in Section 7.2.16.

If a program specifies either DECtape or magtape, the monitor assigns the related unit to the calling job while the call executes. The unit remains assigned after the call completes.

When a program repeatedly executes one of the calls on disk and increments the index for each repetition, the execution time increases for each successive call. The increase occurs because the monitor must read the file name blocks for indices numbered 0 through N-1 before it reads the file name block for index number N. The process is the only one possible since the index value has no other relationship to the actual disk address of the file name block.

When a program repeatedly executes one of the calls on a system disk structure having multiple public disks, the increase in execution time related to the index value is more critical. Since the monitor has no means of determining how many files exist on each unit of a multiple public disk structure, it must read the file name blocks of each unit beginning at unit 0 until the Nth file is read. Therefore, on such a system, execution time can be decreased if the program executes the call repeatedly on each specific unit of the public structure (for example, DK0:, DK1:, and upward) rather than on the entire public structure (SY:).

7.2.17 Monitor Tables And FCB Or DDB Information

The two monitor table SYS system function calls to FIP return to the user program either an address or a data value. They are commonly employed with the PEEK function to read various system parameters and tables which give configuration and run time information. Because it is beyond the scope of this manual to describe the monitor, this section only briefly describes the information returned by the monitor table functions. Section 7.3 describes the use of the PEEK function for certain convenient programming operations.

In this section, each item of information described is denoted by a name in all upper case letters. This name is the same one used to identify the information in the RSTS/E assembly listings. If the name is enclosed by parentheses, the information returned is an address of the data described. If the name is not enclosed by parentheses, the information returned is the actual data value. For example, the get monitor table (part I) call returns NULINE in byte 3. The value returned is the number of terminal lines configured on the system. However, in bytes 11 and 12 is (JOBTBL), the address of the table of jobs. The user program can inspect the address by using the PEEK function.

7.2.17.1 Get Monitor Tables - Part I - Not Privileged (F0=-3)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-3%), the get monitor tables (part I) code
3-30	Not used.

Data Returned:

Byte(s)	Meaning
1-2	Not used
3	NULINE - the number of keyboards configured on the system
4	MAXCNT - the maximum job number allowed during the current time sharing session
5-6	(DEVCNT) - the table of maximum unit numbers for all devices configured on the system
7-8	(DEVPTR) - the table of pointers to device DDBs
9-10	(CORTBL) - the memory allocation table
11-12	(JOBTBL) - the job table
13-14	(JBSTAT) - the job status table
15-16	(JBWAIT) - the table of job wait flags
17-18	(UNTCLU) - the table of unit cluster sizes for mounted disks
19-20	(UNTCNT) - the status table of the error counts and all devices on the system and the count of open files on each device
21-22	(SATCTL) - the table of free block counts for each disk (other than swapping disks) on the system. The table SATCTL contains the least significant word (16 bits) of the double precision unsigned integer (32 bits) count of free blocks. Each word applies to a separate disk unit.
23-24	(TBLNAM) - the program name table
25-26	(SATCTM) - the table of free block counts for each disk (other than swapping disks) on the system. The table SATCTM contains the most significant word (16 bits) of the double precision unsigned integer (32 bit) count of free blocks. Each word applies to a separate disk unit.
27-28	Current date in internal format
29-30	Not used.

Possible Errors: No errors are possible.

7.2.17.2 Get Monitor Tables - Part II - Not Privileged (F0=-12)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-12%), the get monitor tables (part II) code
3-30	Not used.

Data Returned:

Byte(s)	Meaning
1-2	Not used
3-4	(FREES) - the table of free (small and large) buffer information
5-6	(DEVNAM) - the device name table
7-8	(TTILST) - the keyboard input CSR table
9-10	(TTSLST) - the modem default list
11-12	(TTYHCT) - the number of hung terminal errors since system start up
13-14	(JOBCNT) - current number of jobs, current job number limit, configured job number limit
15-16	Reserved
17-18	(ERRCTL) - error logging control data
19-20	(TBLMES) - the table of eligible message receiving jobs
21-22	(LOGNAM) - the table of logical names for mounted disks
23-24	(JOBRES) - the table of job residency pointers
25-26	(CORMAX) - the word containing the size of memory physically present on the system. This value is updated after the RESET command in the TABLE OPTION is executed.
27-30	Not used.

Possible Errors: No errors are possible.

7.2.17.3 Get Open Channel Statistics - Not Privileged (F0=-8)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-8%), the get open channel statistics code
3	CHR$(N%) where N% is the channel number (between 0 and 15) of either the FCB or DDB
4-30	Not used.

Data Returned:

Byte(s)	Meaning
1-2	Not used
3-4	Word 1 of either the FCB or DDB
5-6	Word 2 of either the FCB or DDB
.	.
.	.
.	.
27-28	Word 13 of either the FCB or DDB
29-30	Word 14 of either the FCB or DDB

Possible Errors:

Meaning	ERR Value
I/O CHANNEL NOT OPEN The channel specified in byte 3 of the call is not open.	9

Discussion:

The layout of an FCB and DDB for each device configured on the system is in the listing of the TBL.LST file created during system generation.

The use of this call is rendered obsolete by the STATUS variable described in Section 12.3.5 of the BASIC-PLUS Language Manual.

7.2.18 Enabling and Disabling Disk Cacheing - Privileged (F0=19)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(19%), the enable and disable disk cache code
3	CHR$(N%) where N% is 0 to enable the disk cache and non-zero to disable the disk cache
4-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE	18

An attempt by a non-privileged job to execute this call.

Discussion:

This function enables or disables the disk cache.

7.2.19 Run-Time System Control

A privileged user can conduct time-sharing operations with an auxiliary run-time system supplied by DIGITAL. This section describes SYS System Function Calls -17 and -18, used to build auxiliary run-time systems.

7.2.19.1 Name A Run-Time System - Privileged (F0=-17)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-17%), the name run-time system code
3	CHR$(N%) where N% is the channel number
4-7	Run-time system name in RAD50
8-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
I/O CHANNEL NOT OPEN The channel specified in byte 3 of the call is not open.	9
ILLEGAL SYS() USAGE An attempt by a non-privileged job to execute this call.	18

Discussion:

This SYS function provides the association required for a specially tailored module on the system disk to be accessed as a run-time system. This association is normally made during the system library build procedure and is not required again unless the system is regenerated.

7.2.19.2 Add A Run-Time System - Privileged (F0=-18)

Data passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-18%), the run-time system manipulation code
4-6	Not used.
7-10	Run-time system name in RAD50
11-12	Not used.
13-14	Maximum allowed user image size, in K
15-16	Maximum allowed user image size, in K
17-18	Allowed user image size, in K, at initialization
19-20	Reserved; must be 0
21-22	Not used.
23-24	Device name
25	Unit number
26	Unit number flag
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged job to execute this call; illegal parameters.	18
NO ROOM AVAILABLE FOR FCB A small buffer is not available for FCB.	32
NO RUN-TIME SYSTEM Auxiliary run-time system cannot be found	41

Discussion:

This call adds the appropriate name to the run-time system table in memory. This action is necessary since the initialization code establishes a new table each time system start-up occurs. This call ensures that the appropriate name and parameters are entered in the table.

The maximum and minimum user image sizes of the BASIC run-time system are set to 16K and 2K, respectively. These values are in reference to the user's job swappable image, not the size of the run-time system itself.

Bytes 17 and 18 specify the requested size of the run-time system at initiatlization. If possible, this space is reserved when the run-time system is initialized.

7.2.19.3 Remove A Run-Time System - Privileged (F0=-18)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-18%), the run-time system manipulation code
3	CHR$(4%), remove run-time system
4-6	Not used.
7-10	Run-time system name in RAD50
11-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged job to execute this call; illegal parameters.	18
NO RUN-TIME SYSTEM Auxiliary run-time system cannot be found.	41
ACCOUNT OR DEVICE IN USE Removal of the run-time system is not allowed while it is being used.	3
PROTECTION VIOLATION Removal of the default run-time system is not allowed.	10

Discussion:

The SHUTUP system program automatically performs the remove action when time-sharing operatons are terminated. The module entry is deleted from the run-time system table.

7.2.19.4 Load A Run-Time System - Privileged (F0=-18)

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-18%), the run-time system manipulation code
3	CHR$(2%), load run-time system
4-6	Not used.
7-10	Run-time system name in RAD50
11-12	Loading address
13	128=Permanent residency; 0=Temporary residency
14	1=Read/Write system; 0=Read only system.
15-20	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged job to execute this call; illegal parameters.	18
NO ROOM FOR USER ON DEVICE Run-time system is too large for available storage space.	4
ACCOUNT OR DEVICE IN USE More than one job is attempting to load a run-time system at the same time.	3
NO RUN-TIME SYSTEM Auxiliary run-time system cannot be found.	41

Discussion:

This call loads the run-time system into memory at the 1K section specified. The run-time system is loaded from low memory to high memory at its defined initialized size, if possible. To be loaded without error, enough contiguous user space must be available starting at that location. The location specified in bytes 11 and 12 is the location at which the run-time system is loaded during the current time-sharing session. A -1 value in these bytes specifies the previous load address of this run-time system, if any.

The section of memory chosen must not fragment the user job space to prevent the run-time system from executing a job. For example, assume a system has 24K words of user space available between the 36K and 60K sections of memory. Assume also that a job requires 18K words of user space to run and that the run-time system requires 4K words when resident. If the loading address is 36K, the space between 40K and 60K remains available for an 18K job to run. If the loading address is 42K, however, the user space is fragmented from 36K to 42K and from 46K to 60K. An 18K job area is not available to execute a job using this auxiliary run-time system.

7.2.19.5 Unload A Run-Time System - Privileged (F0=-18)

Data passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-18%), the run-time system manipulation code
3	CHR$(6%), unload run-time system
4-6	Not used.
7-10	Run-time system name in RAD50
11-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS() USAGE An attempt by a non-privileged job to execute this call; illegal parameters.	18
NO RUN-TIME SYSTEM Auxiliary run-time system cannot be found.	41

Discussion:

This call frees the portion of memory occupied by the run-time system. The memory is made available as user job space.

7.3 THE PEEK FUNCTION

The PEEK function allows a privileged user to examine any word location in the monitor part of memory. The user program can examine words in small or large buffers, in the resident portion of the file processor, and in the low core and tables section of memory. The function does not allow a user program to examine the contents of another user's program.

A privileged program executes the PEEK function in the following manner.

```
I% = PEEK(J%)
```

The function takes an (even) integer argument (J%) and returns an integer value (I%). The value returned is the contents of the address in memory specified by the argument. Since, on the PDP-11 computer, addresses of word locations are always even, and odd addresses indicate byte locations, the user must always be careful to specify an even integer address as the argument to PEEK. To examine an odd address, the program must specify the next lower integer as the argument to PEEK. The contents of the odd address is the high order byte of the value returned by PEEK.

The PEEK function is normally used to examine either addresses returned by get monitor tables calls or addresses of fixed monitor locations.

The following are possible errors generated by incorrect usage of the PEEK function.

Meaning	ERR Value
PROTECTION VIOLATION An attempt by a non-privileged user to execute this call	10
UNIBUS TIMEOUT FATAL TRAP The address specified as an argument to PEEK is either odd or out of range of the allowed addresses.	33
MEMORY MANAGEMENT VIOLATION The address specified as an argument to PEEK is illegal (not mapped in the monitor).	35

7.3.1 Fixed Locations In Monitor

The following information is stored in fixed locations in the monitor part of memory and is obtained by executing a PEEK(X%) where X% is the address shown.

Address (decimal)	Name	Meaning
36(word)	IDATE	The date when the system was last started by START
38(word)	ITIME	The time of day when the system was last started by START
512(word)	DATE	Current system date
514(word)	TIME	Current time of day
518(byte)	JOB	Job number times 2 of the job currently running (always is the user's own job number)
520(word)	JOBDA	Address of the job data block (JDB) of the currently running job (always the user's own job data block)

7.3.2 Useful Routines

7.3.2.1 Finding The Current Project-Programmer Number - Two methods exist for a program to determine the project-programmer number under which it is running. The first method, the only one available to non-privileged users, is to execute the read or read and reset accounting data FIP function (F0=14). If the index and the project-programmer number passed in the call are both 0, the project-programmer number returned in bytes 5 and 6 is that of the program executing the call. This first method is slow because it requires FIP handling and possibly requires one or more disk accesses.

The second method, available only to privileged users, is faster and involves executing the PEEK function to examine two bytes in the job data block (JDB) of the job. The contents of the JDB bytes 26 and 27 is the project-programmer number of the current job. The high byte returned by PEEK is the project number; the low byte is the programmer number. The address of the JDB of the currently running job is in the fixed monitor location JOBDA (address 520). The following statement

```
A% = PEEK(PEEK(520%) + 26%)
```

puts the project-programmer word into the variable A%. The following statements put the project number in B% and the programmer number in C%.

```
B% = SWAP%(A%) AND 255%
C% = A% AND 255%
```

7.3.2.2 Determining An ATTACHED Or DETACHED Condition - Only one method exists for a program to determine whether or not it is attached to a terminal. It is beyond the scope of this manual to describe the mechanics of the method. It is sufficient to say that the method determines whether or not a console keyboard exists for the job. The following statements show the procedure.

```
10 IF ((PEEK(PEEK(PEEK(PEEK(520%)))+2%) AND 255%)=(PEEK(518%) AND 255%)
        AND
        (PEEK(PEEK(PEEK(PEEK(520%)))+30%) AND 8192%)=8192%)
        THEN GOTO 20
        ELSE GOTO 30
20 REM: THIS LINE IS REACHED ONLY IF THE JOB IS ATTACHED TO A TERMINAL
25 PRINT "ATTACHED"
        : STOP
30 REM:  THIS LINE IS REACHED ONLY IF THE JOB IS DETACHED
35 STOP
```

Line 10 determines the attached or detached condition. The parentheses are important.

Once a program determines that it is attached to a terminal, it normally is not necessary to find the keyboard number. The program has normal access to the terminal by executing either an OPEN "KB:" statement on a free channel or a PRINT or INPUT statement without a channel specified.

CHAPTER 8

PROGRAMMING CONVENTIONS AND HINTS

Many RSTS/E system programs are designed to run by methods other than by the RUN command. The following are some of the alternative methods.

a. At a logged out terminal by means of LOGIN. For example, SYS typed at a logged out terminal causes the SYSTAT system program to run.

b. At a terminal by means of a CCL command. For example, the standard CCL command QUE runs the QUE system program.

It is useful for the system manager to be able to duplicate some of these actions in his own utility programs on his system. Also, the system manager can alter system programs to tailor them to local installation needs. The guidelines in the following sections describe how to perform these actions.

8.1 RUNNING A PROGRAM FROM A LOGGED OUT TERMINAL

A program runs from a logged out terminal by means of the LOGIN system program. When the user types characters at a terminal not logged in, the monitor runs LOGIN which compares the characters typed with those it is designed to recognize. LOGIN is designed to accept a command line from a logged out terminal and chain to a system program.

The following discussion employs the SYSTAT system program as an example of coding both LOGIN and a user program to run at a logged out terminal. The monitor runs LOGIN at line 32000 if a line is typed at a terminal not logged into the system. LOGIN extracts the characters typed and compares the leftmost characters typed with commands in a set of DATA statements between lines 32200 and 32299. If a command matches the characters and the third item in the DATA statement is 4, LOGIN puts the command line extracted in the core common area and chains to line 32000 of the program and account specified in the second element of the DATA statement. Thus, for SYS typed at a logged out terminal, LOGIN chains to the SYSTAT program in the system library account. The following statement in LOGIN ensures that action.

```
32280 DATA SYS, $SYSTAT, 4
```

To chain to a certain program, the user can supply a DATA statement in the same format between lines 32101 and 32199 in LOGIN. For example, to run a program named HELP under account [2,2] in response to a command named WHAT, insert into LOGIN.BAS a DATA statement similar to the following. The DATA statement must include an account number since there is no default account when running at a terminal not logged into the system.

```
32190 DATA WHAT, "[2,2]HELP", 4
```

After LOGIN is compiled and stored in the system library account, the program HELP under account [2,2] starts running at line 32000 whenever WHAT is typed at a logged out terminal.

The program chained to at line 32000 must contain statements which process the information passed to it in core common as a command line. Provision must also be made for resetting variables used as flags and for initiating error handling. Because a job not logged into the system has no project-programmer number, the program chained to cannot assume a default project-programmer number when opening a file. The CHAIN command in the LOGIN system program does not drop the special login privileges which are afforded by not being logged in. The program to which LOGIN chains can therefore read or write any file on the system because it retains the full privileges. To implement protection, the program itself must perform the protection check. When a logged out job terminates, it can print the BYE message to inform the user that the terminal is still logged out. When the logged out job executes an END statement or a STOP statement, the system immediately removes the program from memory and leaves the user terminal in a logged out state.

8.2 DESIGNING A PROGRAM TO RUN BY A CCL COMMAND

By means of the CCL command option at system generation time, the system manager can specify unique commands which, when typed on a logged in terminal, load and run BASIC-PLUS programs from the system library account. The monitor maintains a table of such commands and programs and, by convention, chains to a program at line 30000.

To pass information to the program, the monitor writes to the core common area the entire line typed at the terminal (after all spaces have been deleted) including the command typed. The user program must extract the data from core common and parse the entire line typed. This action allows the user to design a program to run by several CCL commands. As a convention, programs supplied by DIGITAL and invoked by standard CCL commands reserve lines 30000 to 30999 for routines which parse the command line, check for errors, and dispatch to routines within the program.

8.3 CHANGING LOGIN TO SET A DIFFERENT SWAP MAXIMUM

The LOGIN system program sets the swap maximum to 8K words for all users except those whose project numbers are one. This action means that privileged users run with a swap maximum of 16K words. Since, on many systems, programs must run under non-privileged accounts in job areas larger than 8K words, it is necessary to modify LOGIN to set a swap maximum larger than 8K words.

To modify the LOGIN.BAS program, the user must alter the J% = 8% statement in the first physical line of the multiple statement line at line number 15010 and compile the program on the system library account. The following statement sets the priority, run burst, and swap maximum factors.

```
15010 J% = 8%
      : J% = 16% IF (A% AND -256%) - 256%
      : I$ = SYS(CHR$(6%)+CHR$(-13%)+CHR$(-1%)+
             CHR$(-1%)+CHR$(-2%)+
             CHR$( 0%)+CHR$( 6%)+
             CHR$(-1%)+CHR$( J%))
      : RETURN
```

Change the value 8% in the statement J% = 8% to any value less than or equal to the current default swap maximum used at system start-up time. Re-compile and re-protect the program on the system library account with protection code <60> as follows.

```
COMPILE SY0:LOGIN$
READY
NAME "LOGIN.BAC$" AS "LOGIN.BAC$<232>"
READY
```

It is recommended that the system manager not replace the original source file LOGIN.BAS with the modified version.

8.4 PROGRAMMING HINTS

These sections describe suggested programming methods in two categories: reducing overhead storage space in programs and decreasing required access time for certain operations. Normally, these are mutually exclusive goals. Space is most often saved at the expense of time, and vice versa. In the sections that follow, a discussion of either commodity ignores the other, so it is up to the user to decide when each of these methods can best be used for a particular application. Of course, when both space and time are optimized, as is the case with some of these methods, the entire RSTS/E system as well as the individual program benefits.

8.4.1 Storage Space Overhead

Certain steps can be taken to reduce overhead significantly. This section describes some of the most efficient methods to optimize the storage space available on a RSTS/E system.

Combining statements on a line with the use of colons or backslashes saves statement header space. When using multiple statement lines, remember the maximum line length is 256 characters.

Verbs that always require a statement header, regardless of where they occur, are: DATA, DEF, DIM, FNEND, FOR and NEXT. Whenever possible these statements should be first on a line to reduce statement header overhead.

Statements such as INPUT, which may generate errors, should always be first on a line because a RESUME statement, when executed from an error handling routine, resumes program execution at the nearest preceding statement header. Similarly, since a GOTO statement begins execution only at the first statement of the specified line, the beginning of each routine in a program should be the first statement on a line.

Use the exclamation point (!) within statement lines to indicate remarks. The REM statement or an exclamation point with its own line number requires a 12-byte statement header.

In addition ot the storage space used when referencing a constant or variable, separate space is required to store the value. This space is required for constants every time they are referenced. Once a variable is defined, however, further references generate no more space. For this reason, frequently used constants should be declared as variables. For example:

```
        10 A% = 278%:  B% = 278%
```

requires more storage space than:

```
        10 A% = 278%:  B% = A%
```

And the following statement, accomplishing the same thing, uses the least storage space:

```
10 A%,B% = 278%
```

In some cases assigning a temporary variable saves vector addressing space. For example, consider this program segment:

```
10 FOR I% = 1% TO N%:

     S = S + X(I%):  S2 = S2 + X(I%) * X(I%)

20 NEXT I%
```

Assigning a new variable (T), equal to an indexed variable (X(I%)) decreases the number of bytes of storage area as follows:

```
10 FOR I% = 1% TO N%:

     T = X(I%):

     S = S + T:  S2 = S2 + T * T

20 NEXT I%
```

In addition, the example shown above executes quickly since recalculation of vector addresses is eliminated. Similarly, previously calculated items should be re-used. For example:

```
10 D = SQR(B^2 - 4*A*C)/2%*A:

     PRINT -B/2%*A + SQR(B^2 - 4*A*C)/2%*A;

            -B/2%*A - SQR(B^2 - 4*A*C)/2%*A
```

obviously should be written:

```
10 D = SQR(B^2 - 4*A*C)/2%*A:

     PRINT -B/2%*A + D; -B/2%*A - D
```

On the other hand, certain intermediate terms should be deleted. For example:

```
20 A = B + C:  D = A + E:  F = D + G
```

should be condensed to:

```
20 F = B + C + E + G
```

unless the variables A and D are to be used independently later in the program.

Use integer variables when possible, and always denote them with the percent sign (%). The subscripts in arrays should always be integers, specified with % signs. Use integers in FOR/NEXT loops unless the STEP function value is not an integer. Every constant should include a % or a period (.) each time it is used.

Variables with the same first character should be used when possible. For example:

```
A, A%, A$, A(...), A%(...), A$(...)
```

each have the same first character, but A and B do not, and because of the way BASIC-PLUS stores variables, they use more space. So if the variable R is used in the program, using R% is preferable to using another integer variable.

Always use as few variables as possible; re-use these variables for even more space savings, since no additional overhead is needed.

Variables do not have to be compared to zero explicitly. For example:

```
30 IF M%< >0% THEN 80
```

should be written:

```
30 IF M% THEN 80
```

Space can be saved by using subroutines instead of user-defined functions, but be sure to exit with RETURN (not GOTO) statements.

Individual variable names are often more economical than arrays, since they require less overhead. But if arrays are used, always dimension them first and limit the use of MAT statements.

Use implied FOR loops, which require less core and execute faster, than FOR/NEXT loops. For example:

```
10 FOR I% = 1% TO 10%:

      R% = R%^2%

20 NEXT I%
```

can be written:

```
10 R% = R%^2% FOR i% = 1% to 10%
```

Including the implied FOR loop, above, uses approximately 30% less core and executes faster than the original statements.

Similarly, WHILE and UNITL should be used when possible to replace loops. The statement,

```
10 X% = X% * X%:  IF X%<L% THEN GOTO 10
```

can be written:

```
10 X% = X% * X% WHILE X%<L%
```

When using multiple IF statements, always use the compound IF format. For example:

```
          100 IF X% = W% THEN 250

          110 IF X% = A% THEN 300

          120 IF X% = K% THEN 500

          130 IF X% = L% THEN 600
```

can be rewritten as follows:

```
      100  IF X% = W% THEN 250 ELSE

          IF X% = A% THEN 300 ELSE

          IF X% = K% THEN 500 ELSE

          IF X% = L% THEN 600
```

This compound IF format, above, saves 35% more core than the single IF format.

When a variable is to be compared to some continuous range of values, replace the IF statements with an ON-GOTO statement. For example:

```
      100 IF X% = 4% THEN 250 ELSE

          IF X% = 5% THEN 300 ELSE

          IF X% = 6% THEN 500 ELSE

          IF X% = 7% THEN 600
```

should be replaced with:

```
      100 ON X%-3% GOTO 250,300,500,600
```

A similar technique can be used with strings. The following example:

```
       80 IF A$ = "A" THEN GOTO 100 ELSE

          IF A$ = "B" THEN GOTO 200 ELSE

          IF A$ = "C" THEN GOTO 300 ELSE

          IF A$ = "D" THEN GOTO 400
```

can be rewritten:

```
       80 X% = ASCII(A$)-64%:  ON X% GOTO 100,200,300,400
```

Random string responses can be tested also. For example, to compare A$ with the letters X,K,B and Y, use the following statement:

```
       80 ON INSTR(1%,"XKBY",A$) GOTO 100,200,300,400
```

8.4.2 Decreasing Access Time

There are a number of ways access time can be saved once data or programs are stored on disks. The BASIC-PLUS Language Manual describes some of these methods (for example, see Section 11.5 in that manual).

Methods of setting up files are equally important and are discussed in this section. Most of these methods can be employed by users, but some of them may require the assistance of the system manager.

Open files at the beginning of a program and pre-extend them to their maximum size. Also, pre-allocate scratch data files and, when through using them, simply close them instead of killing them. In this way, they can be re-used without wasting disk space and, ultimately, access time. These techniques tend to reduce directory fragmentation, decreasing access time.

Keep large, frequently used files on separate disks. When two files are often opened at the same time, they too should reside on separate disks. Also, production files and accounts should be kept separate from development and scratch files when possible. If they cannot be kept on separate disks, they should at least be maintained in separate accounts.

File and pack clustersizes should be optimized, as mentioned in the BASIC-PLUS Language Manual and the RSTS/E System Manager's Guide, respectively.

8.4.3 String Manipulation

Pre-extend all strings to their maximum length at the beginning of a program, as follows:

```
10 X$ = SPACE$(L%)
```

where L% is the maximum length of the string. Then use LSET and RSET statements (don't re-use LET) to move data into the string.

The difference between LET and LSET can be seen with the diagrams shown below. Consider the following statements:

```
10 LET A$ = "ABCD"

20 LET C$ = "EFG"
```

$$\begin{tikzcd}
A\$ \arrow[r] & \text{"ABCD"} \\
C\$ \arrow[r] & \text{"EFG"}
\end{tikzcd}$$

When the next statement is executed:

```
30 LET C$ = A$
```

$$\begin{tikzcd}
A\$ \arrow[r] & \text{"ABCD"} \\
C\$ \arrow[ur] &
\end{tikzcd}$$

C$ points to the same part of the I/O buffer as does A$. The old string, "EFG", is wasted space. Note the effect of the LSET statement:

```
40 LSET C$ = A$
```

$$\begin{tikzcd}
A\$ \arrow[r] & \text{"ABCD"} \\
C\$ \arrow[r] & \text{"ABC"}
\end{tikzcd}$$

When a null string is concatenated to A$, the string C$ contains the contents of A$, but no longer points to the I/O buffer:

```
50 LET C$ = A$ + ""
```

$$\begin{tikzcd}
A\$ \arrow[r] & \text{"ABCD"} \\
C\$ \arrow[r] & \text{"ABCD"}
\end{tikzcd}$$

LSET and RSET statements move data; LET statements move pointers. Proper use of LSET and RSET prevents generation of wasted space, and execute faster than LET statements.

The following three algorithms truncate trailing blanks from a data record (for example, a card image). The first two user-defined functions input a string and return the same string without trailing blanks and CR/LF.

The slowest algorithm successively reassigns the argument until it ends with a non-blank:

```
1000 DEF FNT$(X$):

       S$ = LEFT(X$,LEN(X)-1%)

            WHILE RIGHT(X$,LEN(X$)) <= " "

                 AND LEN(X$)>0%

1010      FNT$ = S$

1020 FNEND
```

The following algorithm is much more efficient. It scans backwards until a non-blank character is found. Only one assignment is made.

```
200  DEF FNW$(X$):

       GOTO 2010 IF MID(X$,X%,1%)>" "

            FOR X% = LEN(X$) TO 0% STEP -1%

2010      FNW$ = LEFT(X$,X%)

2020 FNEND
```

The most efficient algorithm uses the data buffer directly, avoiding the assignment caused by the user-defined function. L% is the record length.

```
3000 FOR K% = L% TO 1% STEP -1%:

       FIELD #2%, K%-1% AS L$, 1% AS L$:

       IF L$>" " THEN

            FIELD #2%, K% AS L$: GOTO 3020

3010   NEXT K%: LSET L$ = ""

3020   ! DONE
```

The more efficient algorithms are more cpu-bound because they do less swapping.

APPENDIX A

MAGTAPE LABEL FORMATS

A magtape is said to be in DOS format when the tape labels associated with its files are in DOS format. Similarly, a magtape in ANSI format is one whose labels are in ANSI format. The following sections discuss the differences between DOS and ANSI formats, and how the system handles magtapes written in either format.

A.1 DOS MAGTAPE FORMAT

This section describes the labels and data blocks on a magtape in DOS format, as well as the order in which these items are placed on the tape. For purposes of explanation, assume that the particular magtape under discussion has three files, each containing ten data blocks.

The first part of the magtape is a physical beginning of tape (BOT), a reflective (silver) marker. Immediately past this marker is the first tape label followed, in this case, by ten data blocks and an end of file (EOF) record.

All magtape files begin with a tape label, contain any number of data blocks (default size is 512 bytes per block), and terminate with an end of file record. DOS files may even contain zero data blocks, but a label and end of file record are always required for each file.

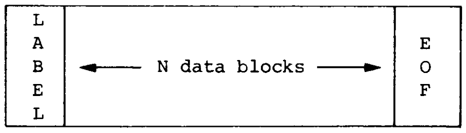


Figure A-1
A DOS Magtape File

Following the first file, another label begins the second file. This label is also followed by ten data blocks and an end of file record. This second file is immediately followed by the third and last file, which consists of a tape label, ten data blocks, and an end of file record. In addition, since the third file on this tape is also the last one, two more end of file records follow. The magtape has three end of file records at this point, signifying a logical end of tape (LEOT).

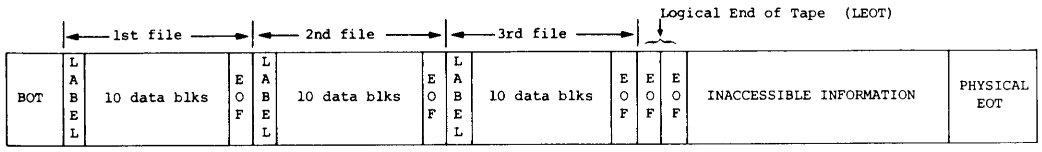


Figure A-2
DOS Magtape Consisting of 3 Files of 10 Data Blocks Apiece

Once the logical end of tape is written on the magtape, it can be written over, but it cannot be read over. Therefore, all information beyond the logical end of tape is inaccessible.

A magtape can contain no files. In that case, three end of file records simply follow the beginning of tape marker.

A.1.1 DOS Labels

The record label which specifies the beginning of a magtape file in DOS format is 14 bytes long. Table A-1 identifies the information contained in each of the record label bytes, numbered from 0 to 13.

Table A-1
DOS Record Label Bytes

Byte	Contents	Data Format
0,1,2,3	File name	2 words in RADIX 50
4,5	File extension	1 word in RADIX 50
6	Programmer number	1 byte in binary
7	Project number	1 byte in binary
8	Protection code	1 byte in binary (always 155(10))
9	Unused	1 byte of zero
10,11	Creation date	1 word in internal date format
12,13	Unused	1 word of zero

The project-programmer number is the account number of the current user, unless some other number is specified in the OPEN statement. If magtapes are to be interchanged with DOS-11 systems, a problem may occur since RSTS-11 treats project-programmer numbers as decimal values, and DOS-11 treats this number (called a UIC under DOS-11) as an octal value. To avoid interchange problems, simply write all files on the tape with a [1,1] project-programmer number, which is the same in both decimal and octal. For example:

```
        100  OPEN "MT0:ABC[1,1]" FOR OUTPUT AS FILE 1%
```

would accomplish this. Note that the project-programmer number is part of the filename string. There could be several files named "ABC" on a tape having different project-programmer numbers associated with them. Often a failure to find a file on a magtape is the result of forgetting to specify the correct account number.

The protection code written by RSTS/E in the DOS label is always 155 decimal (233 octal) which is acceptable to DOS-11. RSTS-11 and DOS-11 use different protection code values. RSTS-11 ignores the value of the protection code when reading the file. This avoids interchange conflicts with DOS-11.

A.2 ANSI MAGTAPE FORMAT

This section describes the labels and data blocks on a magtape in ANSI format. Once again, for purposes of explanation, assume that the particular magtape under discussion has three files, each containing ten data blocks.

The first part of the magtape is a physical beginning of tape reflective marker. The next item on the magtape is a volume label. (A volume is simply a reel of magnetic tape. A volume may contain part of a file, a complete file, or more than one file.)

The first file actually begins with two labels, called header labels (HDR1 and HDR2). These header labels are followed by an end of file (EOF) marker. In this case, ten data blocks are written immediately after the single EOF marker. The data blocks are followed, in order, by another EOF marker, two trailer labels (EOF1 and EOF2), and yet another EOF marker.

All files in ANSI format begin with two header labels. An EOF marker is written on both sides of the data blocks. Two trailer labels follow, and the file is terminated by another EOF marker. When the file is created but no data blocks are written, all the above labels and end of file markers are still present. These labels and end of file markers are always required for each file.

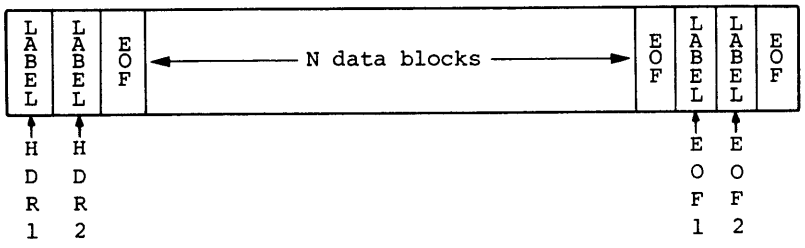


Figure A-3
An ANSI Magtape File

Following the first file, another set of two headers begins the second file. The second file is identical to the first one, consisting of the two header labels, one EOF marker written on each side of ten data blocks, two trailer labels and an EOF marker.

The third file on the tape is identical to the first and second, and is followed by two more EOF markers, signifying a logical end of tape (LEOT).

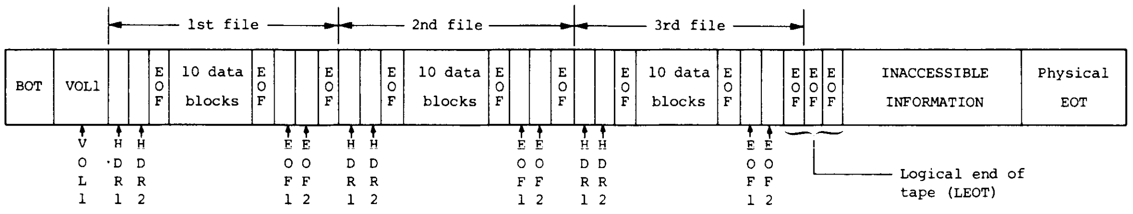


Figure A-4
ANSI Magtape Consisting of 3 Files of 10 Data Blocks Apiece

Once the logical end of tape is written on the magtape, it can be written over, but it cannot be read over. Therefore, all information beyond the logical end of tape is inaccessible.

A magtape must contain at least one complete set of header and trailer labels. In the case where no file exists on the tape (such as a newly zeroed magtape), a dummy file is present with a complete set of labels and end of file records.

A.2.1 ANSI Labels

All ANSI labels are 80 bytes long. Each label can be identified by its first three characters: VOL (volume), HDR (header), and EOF (end of file). The fourth character in each label further defines the sequence of the label within its group. For example, the first and second header labels are HDR1 and HDR2, respectively.

A.2.1.1 Volume Label - This label identifies which volume (reel) of the magtape is being used. Table A-2 shows the character position, field name and contents of each byte (character) in the volume label.

Table A-2
Volume Label Format

Character Position	Field Name and RSTS/E Usage	Contents
1-3	Label Identifier	VOL
4	Label Number	1
5-10	Volume Identifier (Volume label; one to six alphanumerics, blank padded)	1 to 6 alphanumeric characters.
11	Accessibility (RSTS/E writes a space)	A space means no restrictions.
12-37	Reserved	Spaces
38-51	Owner Identifier(1) D%B4431JJJGGG	The contents of this field are used for volume protection.
52-79	Reserved	Spaces
80	Label Standard Version	1

----

(1)The JJJ and GGG in the Owner Identification field represent the user's project and program numbers, respectively. They are written as ASCII digits in decimal notation with leading zeros if needed.

A.2.1.2 Header 1 Label (HDR1) - Table A-3 shows the character position, field name and contents of each byte in the header 1 label.

Table A-3
Header 1 Label Format

Character Position	Field Name and RSTS/E Usage	Contents
1-3	Label Identifier	HDR
4	Label Number	1
5-21	File Identifier (2 to 10 characters FILNAM. or FILNAM.EXT; blank filled)	Any alphanumeric or special character in the ASCII code table
22-27	File-set Identifier (Volume Identifier from the VOL1 label)	Volume ID of first volume in the volume set
28-31	File Section Number (0001)	Numeric characters; starts at 0001
32-35	File Sequence Number	File number within volume set for this file; starts at 0001
36-39	Generation Number (0001)	Numeric characters
40-41	Generation Version (00)	Numeric characters
42-47	Creation Date Today's date in specified format	(SPACE)YYDDD or (SPACE)00000 if no date
48-53	Expiration Date Today's date in specified format	(SPACE)YYDDD or (SPACE)00000 if expired
54	Accessibility	Space
55-60	Block Count	000000
61-73	System Code DECRSTS/E	Name of system which produced the volume. Padded by spaces.
74-80	Reserved	Spaces

A.2.1.3 Header 2 Label (HDR2) - Table A-4 shows the character position, field name and contents of each byte in the header 2 label.

Table A-4
Header 2 Label Format

Character Position	Field Name and RSTS/E Usage	Contents
1-3	Label Identifier	HDR
4	Label Number	2

5	Record Format
	(U is the default)
	(S is unsupported)
		F = Fixed
		D = Variable
		S = Spanned
		U = Undefined

6-10	Block Length (512 is the default)	Numeric Characters Settable by FILESIZE option
11-15	Record Length	Numeric Characters Settable by CLUSTERSIZE option
16-50	System Dependent (M is the default)	Bytes 16-36 (Spaces) Byte 37 = A means first byte of record contains FORTRAN control character = (Space) means LF/CR precedes each record = M means record contains all form control information
51-52	Buffer Offset (00)	Numeric characters 00 on files means PDP-11 produced tapes
53-80	Reserved	Spaces

A.2.1.4 End Of File 1 Label (EOF1) - The EOF1 label is identical to the HDR1 label except for characters 1-3 and 55-60. Table A-5 shows the character position, field name, and contents of each byte in the EOF1 label.

Table A-5
End of File (EOF) 1 Record Format

Character Position	Field Name and RSTS/E Usage	Contents
1-3	Label Identifier	EOF
4	Label Number	1
5-21	File Identifier (2 to 10 characters) FILNAM. or FILNAM.EXT; blank filled)	Any alphanumeric or special character in the ASCII code table
22-27	File-set Identifier (Volume Identifier from the VOL1 label)	Volume ID of first volume in the volume set.
28-31	File Section Number (0001)	Numeric characters: starts at 0001
32-35	File Sequence Number	File number within volume set for this file; starts at 0001
36-39	Generation Number (0001)	Numeric characters
40-41	Generation Version (00)	Numeric characters
42-47	Creation Date Today's date in specified format	(SPACE)YYDDD or (SPACE)00000 if no date
48-53	Expiration Date Today's date in specified format	(SPACE)YYDDD or (SPACE)00000 if expired
54	Accessibility	Space
55-60	Block Count	File size in blocks
61-73	System Code DECRSTS/E	Name of system which produced the volume. Padded by spaces.
74-80	Reserved	Spaces.

A.2.1.5 End Of File 2 Label (EOF2) - The EOF2 label is identical to the HDR2 label except for characters 1-3. Table A-6 shows the character position, field name and contents of each byte in the EOF2 label.

Table A-6
End of File (EOF) 2 Record Format

Character Position	Field Name and RSTS/E Usage	Contents
1-3	Label Identifier	EOF
4	Label Number	2

5	Record Format
	(U is the default)
	(S is unsupported)
		F = Fixed
		D = Variable
		S = Spanned
		U = Variable

6-10	Block Length (512 is the default)	Numeric Characters Settable by FILESIZE option
11-15	Record Length	Numeric Characters Settable by CLUSTERSIZE option
16-50	System Dependent (M is the default)	Bytes 16-36 (Spaces) Byte 37 = A means first byte of record contains FORTRAN control character = (Space) means LF/CR precedes each record = M means record contains all form control information
51-52	Buffer Offset (00)	Numeric characters 00 on files mean PDP-11 produced tapes
53-80	Reserved	Spaces

A.3 ZEROING MAGTAPES

This section describes how a magtape is zeroed. See Section 4.18.7 of the RSTS-11 System User's Guide for the procedure for zeroing magtapes via the Peripheral Interchange Program (PIP).

A magtape written in DOS format is zeroed in the following manner:

1. Magtape is rewound.
2. Three end of file (EOF) records are written on the tape.
3. Magtape is again rewound.

A magtape written in ANSI format is zeroed in the following manner:

1. Magtape is rewound.
2. A volume label (VOL1) is written on the tape. The volume identifier is in bytes 7 through 10, in RAD50.
3. Two header labels (HDR1 and HDR2) are written on the tape.
4. Two end of file (EOF) records are written on the tape.
5. Two trailer labels (EOF1 and EOF2) are written on the tape.
6. Three end of file (EOF) records are written on the tape.
7. Magtape is again rewound.

Notice that in the case of magtapes written in ANSI format, the two header labels (HDR1 and HDR2), two EOF records, two trailer labels (EOF1 and EOF2) and a final EOF record comprise a dummy file. Also, in both the DOS format and the ANSI format case, three EOF records are the last items written on the tape. These three EOF records form the logical end of tape (LEOT).

APPENDIX B

CARD CODES

The RSTS card driver can be configured for one of three different punched card codes. These are DEC029 codes, 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.

CHARACTER	ASCII$_{1Ø}$	DECØ29	DECØ26	14Ø1
{	123	12 Ø	12 Ø	UNUSED
}	125	11 Ø	11 Ø	UNUSED
SPACE	32	NONE	NONE	NONE
!	33	12 8 7	12 8 7	11 Ø
"	34	8 7	Ø 8 5	Ø 8 2
#	35	8 3	Ø 8 6	8 3
$	36	11 8 3	11 8 3	11 8 3
%	37	Ø 8 4	Ø 8 7	Ø 8 4
&	38	12	11 8 7	12
'	39	8 5	8 6	12 8 4
(	4Ø	12 8 5	Ø 8 4	8 7
)	41	11 8 5	12 8 4	Ø 8 7
*	42	11 8 4	11 8 4	11 8 4
+	43	12 8 6	12	Ø 8 5
,	44	Ø 8 3	Ø 8 3	Ø 8 3
-	45	11	11	11
.	46	12 8 3	12 8 3	12 8 3
/	47	Ø 1	Ø 1	Ø 1
Ø	48	Ø	Ø	Ø
1	49	1	1	1
2	5Ø	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	Ø 8 2	11 8 6
<	6Ø	12 8 4	12 8 6	12 8 6
=	61	8 6	8 3	11 8 7
>	62	Ø 8 6	11 8 6	8 6
?	63	Ø 8 7	12 8 2	12 Ø

CHARACTER	ASCII$_{1Ø}$	DECØ29	DECØ26	14Ø1
@	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	7Ø	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	8Ø	11 7	11 7	11 7
Q	81	11 8	11 8	11 8
R	82	11 9	11 9	11 9
S	83	Ø 2	Ø 2	Ø 2
T	84	Ø 3	Ø 3	Ø 3
U	85	Ø 4	Ø 4	Ø 4
V	86	Ø 5	Ø 5	Ø 5
W	87	Ø 6	Ø 6	Ø 6
X	88	Ø 7	Ø 7	Ø 7
Y	89	Ø 8	Ø 8	Ø 8
Z	9Ø	Ø 9	Ø 9	Ø 9
[	91	12 8 2	11 8 5	12 8 5
\	92	Ø 8 2	8 7	Ø 8 6
]	93	11 8 2	12 8 5	11 8 5
↑ or ^	94	11 8 7	8 5	unused
← or _	95	Ø 8 5	8 2	12 8 7

EOF is 12-11-Ø-1 punch or a 12-11-Ø-1-6-7-8-9 punch.

APPENDIX C

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 character/position 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
```

Radix-5Ø Character/Position Table

Single Char. or First Char.		Second Character		Third Character	
A	ØØ31ØØ	A	ØØØØ5Ø	A	ØØØØØ1
B	ØØ62ØØ	B	ØØØ12Ø	B	ØØØØØ2
C	Ø113ØØ	C	ØØØ17Ø	C	ØØØØØ3
D	Ø144ØØ	D	ØØØ24Ø	D	ØØØØØ4
E	Ø175ØØ	E	ØØØ31Ø	E	ØØØØØ5
F	Ø226ØØ	F	ØØØ36Ø	F	ØØØØØ6
G	Ø257ØØ	G	ØØØ43Ø	G	ØØØØØ7
H	Ø31ØØØ	H	ØØØ5ØØ	H	ØØØØ1Ø
I	Ø341ØØ	I	ØØØ55Ø	I	ØØØØ11
J	Ø372ØØ	J	ØØØ62Ø	J	ØØØØ12
K	Ø423ØØ	K	ØØØ67Ø	K	ØØØØ13
L	Ø454ØØ	L	ØØØ74Ø	L	ØØØØ14
M	Ø5Ø5ØØ	M	ØØ1Ø1Ø	M	ØØØØ15
N	Ø536ØØ	N	ØØ1Ø6Ø	N	ØØØØ16
O	Ø567ØØ	O	ØØ113Ø	O	ØØØØ17
P	Ø62ØØØ	P	ØØ12ØØ	P	ØØØØ2Ø
Q	Ø651ØØ	Q	ØØ125Ø	Q	ØØØØ21
R	Ø7Ø2ØØ	R	ØØ132Ø	R	ØØØØ22
S	Ø733ØØ	S	ØØ137Ø	S	ØØØØ23
T	Ø764ØØ	T	ØØ144Ø	T	ØØØØ24
U	1Ø15ØØ	U	ØØ151Ø	U	ØØØØ25
V	1Ø46ØØ	V	ØØ156Ø	V	ØØØØ26
W	1Ø77ØØ	W	ØØ163Ø	W	ØØØØ27
X	113ØØØ	X	ØØ17ØØ	X	ØØØØ3Ø
Y	1161ØØ	Y	ØØ175Ø	Y	ØØØØ31
Z	1212ØØ	Z	ØØ2Ø2Ø	Z	ØØØØ32
$	1243ØØ	$	ØØ2Ø7Ø	$	ØØØØ33
.	1274ØØ	.	ØØ214Ø	.	ØØØØ34
unused	1325ØØ	unused	ØØ221Ø	unused	ØØØØ35
Ø	1356ØØ	Ø	ØØ226Ø	Ø	ØØØØ36
1	14Ø7ØØ	1	ØØ233Ø	1	ØØØØ37
2	144ØØØ	2	ØØ24ØØ	2	ØØØØ4Ø
3	1471ØØ	3	ØØ245Ø	3	ØØØØ41
4	1522ØØ	4	ØØ252Ø	4	ØØØØ42
5	1553ØØ	5	ØØ257Ø	5	ØØØØ43
6	16Ø4ØØ	6	ØØ264Ø	6	ØØØØ44
7	1635ØØ	7	ØØ271Ø	7	ØØØØ45
8	1666ØØ	8	ØØ276Ø	8	ØØØØ46
9	1717ØØ	9	ØØ3Ø3Ø	9	ØØØØ47

APPENDIX D

ASCII CHARACTER CODES

Decimal Value	ASCII Character	RSTS Usage
Ø	NUL	FILL character
1	SOH	
2	STX	
3	ETX	CTRL/C
4	EOT	
5	ENQ	
6	ACK	
7	BEL	BELL
8	BS	
9	HT	HORIZONTAL TAB
1Ø	LF	LINE FEED
11	VT	VERTICAL TAB
12	FF	FORM FEED
13	CR	CARRIAGE RETURN
14	SO	
15	SI	CTRL/O
16	DLE	
17	DC1	
18	DC2	
19	DC3	
2Ø	DC4	
21	NAK	CTRL/U
22	SYN	
23	ETB	
24	CAN	
25	EM	
26	SUB	CTRL/Z
27	ESC	ESCAPE[1]
28	FS	
29	GS	
3Ø	RS	
31	US	
32	SP	SPACE
33	!	
34	"	
35	#	
36	$	
37	%	
38	&	
39	'	APOSTROPHE
4Ø	(	
41	)	
42	*	
43	+	
44	,	COMMA
45	-	
46	.	
47	/	
48	Ø	
49	1	
5Ø	2	
51	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	
58	:	
59	;	
6Ø	<	
61	=	
62	>	
63	?	
64	@	
65	A	
66	B	
67	C	
68	D	
69	E	
7Ø	F	
71	G	
72	H	
73	I	
74	J	
75	K	
76	L	
77	M	
78	N	
79	O	
8Ø	P	
81	Q	
82	R	
83	S	
84	T	
85	U	
86	V	
87	W	
88	X	
89	Y	
9Ø	Z	
91	[	
92	\	Backslash
93	]	
94	^	or ↑
95	_	or ←
96	`	Grave accent
97	a	
98	b	
99	c	
1ØØ	d	
1Ø1	e	
1Ø2	f	
1Ø3	g	
1Ø4	h	
1Ø5	i	
1Ø6	j	
1Ø7	k	
1Ø8	l	
1Ø9	m	
11Ø	n	
111	o	
112	p	
113	q	
114	r	
115	s	
116	t	
117	u	
118	v	
119	w	
12Ø	x	
121	y	
122	z	
123	{	
124	|	Vertical Line
125	}	
126	~	Tilde
127	DEL	RUBOUT

[1]ALTMODE (ASCII 125) or PREFIX (ASCII 126) keys which appear on some terminals are translated internally into ESCAPE.

APPENDIX E

ERROR MESSAGES

E.1 USER RECOVERABLE

A (C) in the description of the error message indicates that program execution continues, following printing of the error message, if an ON ERROR GOTO statement is not present. Normally, execution terminates on an error condition, the error message is printed, and the system prints READY. The ERR column gives the value of the ERR variable (see Section 8.4, BASIC-PLUS Language Manual).

ERR	Message Printed	Meaning
1	BAD DIRECTORY FOR DEVICE	The directory of the device referenced is in an unreadable format.
2	ILLEGAL FILE NAME	The filename specified is not acceptable. It contains unacceptable characters or the filename specification format has been violated.
3	ACCOUNT OR DEVICE IN USE	Removal or dismounting of the account or device cannot be done since one or more users are currently using it.
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.
6	NOT A VALID DEVICE	Attempt to use an illegal or nonexistent device specification.
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 device requested is currently reserved by another user.
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. (SPR)
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.
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 ID'S 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.
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	ON ERROR-GOTO subroutine was entered through a program trapped CTRL/C. See a description of the SYS system function.
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.
32	NO ROOM AVAILABLE FOR FCB	When the user accesses a file under programmed control in RSTS-11, a system control structure called an

		FCB requires one small buffer and one small buffer is not available for the FCB.
33	UNIBUS TIMEOUT FATAL TRAP	This hardware error occurs when an attempt is made to address nonexistent 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.
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. (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.
38	MEMORY PARITY FAILURE	A parity error was detected in the memory occupied by this job.
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	NO RUN-TIME SYSTEM	Reserved.
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)
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 -32767<n<32767. If no transfer to an error handling routine is made, zero is returned as the integer value. (C)
52	ILLEGAL NUMBER	Integer or floating point overflow or underflow.
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 -32766.
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)

E.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 comand 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.
CATASTROPHIC ERROR	The user program data structures are destroyed. This normally indicates a BASIC-PLUS This normally indicates a BASIC-PLUS malfunction and, if reproducible, should be reported to DEC on a Software Performance Report form (SPR).
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.
EXPESSION TOO COMPLICATED	This error usually occurs when parentheses have been nested too deeply. The depth allowable is dependent on the individual expression.
FIELD OVERFLOWS BUFFER	Attempt to use FIELD to allocate more space than exists in the specified buffer.

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 RENAME followed by SAVE.
FNEND WITHOUT DEF	An FNEND statement was encountered in the user program without a previous function call having been executed.
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 MAGTAPE() USAGE	Improper use of the MAGTAPE function.
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 being redefined in a manner inconsistent in the number of type of arguments with one or more calls to that function existing in the program.
INCONSISTENT SUBSCRIPT USE	A subscripted variable is being used with a different number of dimensions from the number with which it was originally defined.
K OF CORE USED	Message printed by the LENGTH command, preceded by the appropriate number describing the user program currently in core to the nearest 1K.
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 CORE EXCEEDED	User program grew to be too large to run or compile in the area of core assigned to each user at the given installation.
MISSING SPECIAL FEATURE	User program employs a BASIC-PLUS feature not present on the given installation.
MODIFIER ERROR	Attempt to use one of the statement modifiers (FOR, WHILE, UNTIL, IF, or UNLESS) incorrectly.
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 A RANDOM ACCESS DEVICE	Attempt to perform random access I/O to a non-random access device.
NOT ENOUGH AVAILABLE CORE	The already compiled user program is too large to run in the area of core assigned to each user at the given installation.
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	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.
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 the user program without a previous

	GOSUB statement having been executed.
SCALE FACTOR INTERLOCK	An attempt was made to execute a program or source statement after changing the current scale factor. Use REPLACE and OLD or use RUN to execute without error.
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.
TEXT TRUNCATED	No BASIC-PLUS statement can be more than 255 characters long.
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 CALL	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 with an incompatible version of RSTS. Program source must be recompiled.

INDEX

RSTS/E Programming
Manual
DEC-11-ORPMA-A-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (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 ________

If you require a written reply, please check here. ☐

Please cut along this line.

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754