pdp11

RSTS/E Programming Manual

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

digital

RSTS/E
Programming Manual

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

digital equipment corporation · maynard. massachusetts

First Printing, July 1975
Revised: November 1976

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

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

FIGURES

TABLES

CONTENTS (Cont.)

TABLES (Cont.)

PREFACE

This manual describes RSTS/E special programming techniques, many of which are not available prior to V06B-02 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 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/E System User's Guide,* in addition to a thorough knowledge of the BASIC-PLUS language. For this reason, Senior, Application and System 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.*

NOTE

All examples in the manual are written to execute in BASIC-PLUS no extend mode unless otherwise noted.

CHAPTER 1
DISKS

1.1 SYSTEM ACCOUNTS

RSTS/E systems have three accounts which are essential to the operation of the system. The Master File Directory (MFD) account [1,1] is used on all disk devices in the system to control access. It is described in Sections 1.1.1 and 1.1.2. The system library account [1,2] is required on system disks for use 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 1.1.3. 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 Sections 1.1.4 and 1.1.5.

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.

Using SYS 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.

1.1.2 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. Disk devices over and above the system disk are called non-system disks. 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 non-system disk initially contains UFD information for accounts [0,1] and [1,1] only. The MFD thereafter contains account and storage information for that device only.

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

Type	Description	Explanation
Identification	Project-programmer number (account)	Refer to the *RSTS/E System User's Guide.*
	Password	1 to 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)	Memory 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.

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 nonprivileged user desires to create a file on a private pack, he cannot do so unless his account number already exists 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. The pack id is utilized whenever the disk is logically mounted via the system program UMOUNT, UTILTY or INIT and whenever the disk is logically dismounted.

[1] 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 before it is physically dismounted to avoid corruption of the file structures.

1.1.3 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 *RSTS/E System Generation Manual.*

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 INIT system program stored in account [1,2] is executed automatically. This program executes commands in the file START.CTL or in another command file. 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 without operator intervention.

1.1.4 System Account [0,1]

The system account [0,1] is created by the DSKINT initialization option. DSKINT creates two files required for all RSTS/E disks: the storage allocation file SATT.SYS and the bad block file BADB.SYS.

Account [0,1] on the system disk contains files used for system operation. A minimum set of these files is created during the system generation procedure. The set consists of the following four files.

1. INIT.SYS, the system initialization code.
2. A file with extension SIL, the monitor code.
3. A file with extension RTS, the run-time system code.
4. A file with extension ERR, the error message text.

The REFRESH initialization option creates other files in account [0,1] on both the system and non-system disks. REFRESH can also locate these files on a disk and can mark them as nondeletable. These files are described in the following sections.

1.1.4.1 Allocating Disk Storage Space – The file SATT.SYS is the mechanism by which RSTS/E controls the allocation and deallocation of storage space for a 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, determines 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.

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 minimum number of contiguous 256-word blocks a cluster comprises on a specific disk 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. The minimum value for a pack cluster size is the device cluster size for the disk type. 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; for an RP04 and RP05 type disk, 4, 8 or 16; and for an RP06 disk, 8 or 16.

Table 1-2 Valid Cluster Size Ranges

Cluster Size	Minimum Size	Maximum Size (decimal)	When Defined
Pack (for non-system disk, public and private)	1 (RF11/RK05 RK06) 2 RP02/RP03) 4 RP04/RP05 8 (RP06)	16	At initialization time via DSKINT option (stored in MFD)
Directory (both for MFD and UFD)	Pack Cluster Size	16	At creation of the directory via either the DSKINT initialization option, REACT, or SYS system function.
File	Pack Cluster Size	256	At creation of the file via either an OPEN or OPEN FOR OUTPUT.

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 the 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.

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

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 and have no attributes.)

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

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.

1.1.4.2 Bad Block File – The bad block file BADB.SYS is the mechanism by which the system manager removes from use unreliable storage on system and non-system disks. BADB.SYS is created in account [0,1] by the DSKINT initialization option. DSKINT can thoroughly check each block on a disk for reliability. If any block on a disk pack or cartridge is faulty, DSKINT allocates the cluster in which the bad block resides to the file BADB.SYS. The bad block file, therefore, contains no data but merely removes from use those clusters found to contain unreliable blocks.

As a disk is exercised during time sharing operations, additional unreliable portions of a disk may be uncovered. By checking the data errors recorded in the system error log, the system manager can isolate these bad blocks. Through the REFRESH initialization option, the manager can add newly discovered bad blocks to BADB.SYS. Once a bad block is allocated to BADB.SYS, it can not be deallocated.

1.1.4.3 System Overlay File and DECtape Directory – Certain monitor code is not resident in memory but resides on disk. This code is loaded into memory on a demand basis and overlays a certain part of the monitor. The overlay code is normally contained in the monitor save image library (SIL). Optimum efficiency is gained when this code resides 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 to create a separate, contiguous file which will contain the overlay code. The file can then be placed on a fixed-head disk or optimally positioned on a fast, moving head disk. The standard name of this separate file is OVR.SYS. At the start of time sharing operations, the system manager can cause the overlay file to be added to the system. Thereafter, the system accesses the copy of the overlay code in the optimally positioned file rather than in the original code in the SIL.

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 optimally positioned on the moving head system device.

1.1.4.4 System Operations File – All monitor resident and nonresident code for time sharing operations resides in account [0,1] on the system disk. This file is structured in save image library format and must have an extension .SIL. Multiple monitor files can reside on the system disk but only one such file is installed at a time. The installed monitor file is marked as nondeletable and is the one which is bootstrapped from disk when time sharing operations are started.

1.1.4.5 Error Messages File – Distributed with each RSTS/E system is a file containing the BASIC-PLUS error messages. This file is called ERR.ERR and must exist in account [0,1] on the system disk. Optimum efficiency is gained when this code resides on a fast access, fixed head disk. This is the file usually established as the default error message file.

The REFRESH option allows the system manager to create a separate contiguous file and position it on a moving head disk. The standard name for this file is ERR.SYS. At the start of time sharing operations, the system manager can cause this separate file to be added as the error message file on the system. The monitor copies the contents of the established default error message file to this optimally positioned file. Thereafter, the system accesses the copy in the optimally positioned file instead of the established default file.

1.1.4.6 Saving Information After a Crash – The system crash file CRASH.SYS contains a dump of the read/write area of the monitor at the time of a system crash. The file is optional, but if it exists, it must reside on the system disk. The size of this file depends on the size of the monitor read/write area. The monitor read/write area size varies according to hardware and software configuration but is usually between 32 and 64 blocks. To reserve additional, contiguous space, the system manager can create CRASH.SYS at a size larger than required. In no case, however, can CRASH.SYS be larger than 80 blocks.

1.1.4.7 Run-Time System Files – The account [0,1] on the system disk must contain one file with an extension .RTS. This file is the system default run-time system and is automatically loaded into memory by the monitor at the start of time sharing operations. The default run-time system must reside on the system disk because that disk is the only one mounted at system start up time.

The DEFAULT initialization option establishes which file in account [0,1] on the system disk is the default RTS. The option ensures that the file is in the correct format, that it is contiguous and that it is not deletable. Because more than one RTS file can exist in account [0,1], this default setting tells the monitor which file to load and what memory size is required.

The system manager may add, as auxiliary run-time systems, other files with extensions of .RTS in account [0,1] on either the system disk or non-system disks.

All run-time system files must occupy contiguous space on disk. This condition allows a run-time system to be loaded into memory as fast as possible.

1.1.4.8 Initialization Code – The system initialization code is stored in the file INIT.SYS on account [0,1] on the system disk. When the system disk is bootstrapped, a secondary bootstrap loads the main part of the initialization code into memory. The initialization code is a large, stand-alone program. It performs consistency checks on system software and hardware. It allows the system manager to initialize and format disks, to install patches, to enable and disable device controllers and units, to manipulate files in account [0,1] on both system and non-system disk, to set default time sharing characteristics, and to set characteristics of certain hardware units. An important feature of the initialization code is that it allows bad blocks to be added to the bad block file BADB.SYS in account [0,1].

At the start of time sharing, the RSTS/E monitor code replaces the initialization code in memory.

1.1.4.9 Swapping Storage – Nonresident jobs on RSTS/E are kept in predefined areas on disk called swapping storage. RSTS/E provides four distinct areas which are referred to as swapping slots. The slots are numbered 0 through 3. The swapping slot numbered 2 is required on all systems; the other slots are optional. Slot number 2 must reside on the system disk and must be called SWAP.SYS.

The REFRESH initialization option creates SWAP.SYS in account [0,1] on the system disk and allows the system manager to create other files in account [0,1] to be used as swapping slots 0, 1, and 3. These other files are optional and may have any names the system manager chooses. The names recommended are SWAP0.SYS, SWAP1.SYS and SWAP3.SYS for slots 0, 1, and 3.

RSTS/E uses the various swapping slots in a predefined manner. A highly interactive job which must be removed from memory is stored in the lowest numbered slot available. The system searches for an empty space starting at the lowest numbered active slot. Conversely, a job with infrequent activity is stored in the highest numbered slot available. Such relatively inactive jobs are those which sleep most of the time until an event occurs. Examples are the system error logging program ERRCPY and the operator services and spooling programs OPSER, SPOOL, QUEMAN and BATCH.

A swapping slot can be either a file or an entire device. By having an entire device as a swapping slot, the system manager can take advantage of high speed, fixed head disks on the system. The manager can add, as the lowest numbered slots, RS03 or RS04 disk units at system start up. As a result, the system will swap the highly interactive jobs to the fastest device on the system. If a file on a moving-head disk device is to be used as a swapping slot, the system manager can position the file in the middle of the disk to minimize the time required for positioning the read/write heads. On systems with multiple moving head disks, two files can be positioned on separate drives to take advantage of overlapped seeks.

A swapping slot other than slot 2 is dynamic. The system manager adds slots at the start of time sharing to allow the maximum number of jobs to run. During time sharing a swapping slot can be removed and added again as another device or file. Dynamic adding and removing of swapping slots allows time sharing to continue when hardware problems on a device being used for swapping would normally require discontinuing system operation.

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

The system account [0,1] on a non-system disk originally contains two required files, SATT.SYS and BADB.SYS. The DSKINT initialization option creates these files similarly for non-system disks as for the system disk. Account [0,1] on a non-system disk, either public or private, can contain other optional system files.

The REFRESH initialization option can be used on a non-system disk as well as on the system disk to manipulate system files in account [0,1]. REFRESH adds blocks to the BADB.SYS file. It can create and position contiguous files (such as a swapping file or the overlay file) on a non-system disk. REFRESH can also mark files in account [0,1] as nondeletable. Non-system disks may contain auxiliary run-time system files.

1.2 PRIVILEGE

Privilege is a special condition for a user job. With privilege, a job has capabilities not available to other, non-privileged jobs on the system. The following sections define privilege and explain how privilege is attained and relinquished.

1.2.1 Privileged Capabilities

A privileged job on RSTS/E has the following special capabilities.

1. unlimited access on the system,
2. ability to designate privileged programs,
3. use of privileged aspects of system programs, and
4. use of privileged SYS system functions and the PEEK function.

Unlimited access means that no protection code can protect a file against read and write access. A privileged job can create and delete files under any account number and can access files on LOCKED disks. Such unlimited access by a privileged job does not generate the normal PROTECTION VIOLATION error.

NOTE

Privilege does not bypass intrinsic protection mechanisms such as locked blocks in updating files.

A privileged job can set the privileged bit in the protection code. The privileged bit, with the compiled file protection bit, designates a program as privileged. Thus, any program with a protection code of <192> (the sum of the privileged protection <128> and the compiled file protection <64>) or greater denotes a privileged program. Both protection code values must be set for a program to have privilege.

When a job has privilege, it can use privileged aspects of system programs, privileged SYS system functions and the PEEK function.

Many system programs are designated privileged at system generation time because they execute privileged SYS system functions and the PEEK function. Because some system programs execute privileged operations, the related programs and operations are described in manuals separate from those the beginning user normally has access to. The *RSTS/E System Manager's Guide* describes privileged system programs and privileged aspects of system programs. This manual describes the privileged SYS system functions and the PEEK function.

1.2.2 Privilege and System Operation

A job has privilege under one of the following conditions:

1. it is a logged-out job (a job without an account),
2. it is running under a privileged account, or
3. it is running a privileged program.

The privilege remains or is dropped depending on how processing continues for the job.

A logged-out job has privilege because the system must minimally perform certain privileged operations to log a job into the system. The privilege remains in effect as long as the job remains logged-out.

A job running under a privileged account has privilege. A privileged account is defined as one whose project number is 1. The account number thus has the form [1,*] where the asterisk (*) represents a programmer number between 0 and 254. (The privileged account with a programmer number of 1 is the master file directory for disks.) The system library account [1,2] is an example of a privileged account.

A job running under a privileged account has permanent privilege. The privilege remains in effect until the job is logged out or the job changes to a nonprivileged account (an account whose project number is not one).

The NAME AS statement is the only method of setting the privileged bit in the protection code. Permanent privilege is required to set the privileged bit. To designate a privileged program, the privileged job must assign the protection code of <192> or greater to the compiled form of the program. The following sample statement shows the procedure.

```
10 NAME 'FILE.BAC' AS 'FILE.BAC<232>'
```

The resultant protection code designates the privileged and compiled protection codes (<128> and <64>) with read and write protection against the owner's project (<8>) and read and write protection against all other's not in the owner's project (<32>). This protection code combination enables users within and without the owner's project to run the privileged program but prohibits those users from reading and writing (and therefore being able to delete or rename) the file.

The NAME AS statement can also be used in BASIC-PLUS immediate mode to designate a compiled file as privileged. The only restriction, however, is that the job under which the statement is executed must itself have permanent privilege.

A job running a privileged program has temporary privilege unless it is running under an account which has permanent privilege. The job gains the temporary privilege when it runs a file whose privilege bit is set. The privilege remains until the privileged program exits or until the program drops its temporary privilege. This type of privilege is necessarily temporary because users of both privileged and nonprivileged accounts may be able to run a privileged program. Accordingly, if the privilege were not temporary, any unexpected halt in the job would leave the system vulnerable to unwarranted tampering. A job running permanently privileged, however, can never drop its privilege.

A temporarily privileged job can rely on the normal protection mechanisms built into the system. Under programmed control, the program can either permanently or temporarily relinquish (drop) its temporary privilege by executing a SYS system function call. This ability to drop privilege allows a job to selectively perform privileged operations. For example, a job could initially set itself up using privileged capabilities and then permanently drop its privilege because further processing does not require privilege. Alternatively, a job could selectively drop and later regain its temporary privilege depending upon the type of processing required. Regaining temporary privilege is accomplished by executing the same SYS system function which allows the job to drop privilege. Regaining privilege, however, is possible only if the job temporarily dropped privilege rather than permanently dropped privilege.

Under certain conditions, the run-time system controls whether temporary privilege is retained. When a program runs via CCL command at other than the lowest numbered line, BASIC-PLUS may drop a job's temporary privilege. The conditions under which this action occurs are described in Section 9.2.4. When a program is run by a CHAIN statement at a line number not the lowest numbered line, BASIC-PLUS unconditionally drops a job's temporary privilege.

1.2.3 Guidelines for Privileged Operation

Permanent privilege gives to a user the full capability of the RSTS/E system but also requires of the user a distinct responsibility. It must be emphasized that these capabilities can destroy the system. No error messages are generated because privilege bypasses the normal protection mechanisms. To avoid destroying data, privilege must be carefully controlled and used.

Privilege capability begins with the assignment of privileged accounts. At system generation time, two privileged accounts, the system library [1,2] and the MFD account [1,1], are created. By having access to the system library, the system manager has privilege capability. One such capability is the ability to designate other users as privileged by assigning them accounts with a project number of 1. Account [1,1] can be used as a privileged account but such use is strongly discouraged.

Users with privileged accounts have all the capabilities of the system manager. Specifically, privileged users can design and implement routines which use privileged SYS system functions and the PEEK function. Privileged users can also designate programs as privileged.

If a program is designated as privileged and is not protected against execution, any user can run the program with temporary privilege. This action extends the use of privileged functions to nonprivileged users in the same way standard system library programs do. For example, the program TTYSET allows a user to change characteristics of his terminal. Such an operation is privileged. With temporary privilege, however, TTYSET executes the normally privileged operation for an owner of a nonprivileged account. TTYSET does not generate the expected protection violation error.

The same TTYSET program additionally allows a privileged user to change characteristics of a terminal other than his own. But TTYSET is coded to ensure that a user attempting to change the characteristics of a terminal other than his own is indeed a permanently privileged user. As a result, TTYSET makes available to a nonprivileged user certain privileged operations but restricts other privileged features to privileged users.

1.3 NON-FILE STRUCTURED DISK OPERATION

Non-file structured disk operation permits the user program to access specific blocks on a disk. This capability enables the user to process non-RSTS/E file structured disks under RSTS/E. (Extended PIP performs directory and transfer operations on DOS structured disks, but permits no write operations. See *RSTS/E System User's Guide*, Section 4.18.2.)

1.3.1 Opening a Disk for Nonfile Structured Processing

To initiate nonfile structured processing, the program specifies only a device designator in the OPEN statement. Only the OPEN and OPEN FOR INPUT statements are valid. The following two sample statements are equivalent:

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

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

Both allow reading and writing of physical blocks on RK unit 1. A third sample statement results in the error DISK PACK IS NOT MOUNTED.

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

To prevent other programs from accessing a nonfile structured disk, a privileged program can assign the device.

1.3.2 Accessing a Block

Before writing a program that accesses specific disk blocks, the user needs to know the terms logical block, device cluster, device cluster size, and device cluster number.

A logical block is 256 words of disk data. Logical blocks are numbered starting at 0 so that logical block n is bordered by logical blocks n+1 and n-1.

A group of contiguous logical blocks forms a device cluster. The device cluster size is the number of logical blocks in the group. The device cluster size is fixed for each type of disk; it may be 1, 2, 4, 8, or 16. The device cluster size represents the minimum amount of information (the minimum number of logical blocks) that can be retrieved or written in one nonfile structured I/O operation. Device clusters are numbered from 0 to the maximum shown in Table 1-3.

Table 1-3 Nonfile Structured Disk Default Characteristics

Device	Device Cluster Size	Default Buffer Size (in Bytes)	Maximum RECORD Number
RC11	1	512	(number of platters *256)-1
RF11	1	512	(number of platters *1024)-1
RS03	1	512	1023
RS04	1	512	2047
RK05,RK05F	1	512	4799
RK06	1	512	27103
RP02	2	1024	19999
RP03	2	1024	39999
RP04,RP05	4	2048	41799
RP06	8	4096	41799

After the user program opens a disk device for nonfile structured processing, he can use the RECORD option in GET and PUT statements to read and write specific logical blocks on the disk. The record number that the user specifies designates a device cluster number. Thus, on an RK05, RECORD 4100% refers to logical block 4100 on the disk, because the device cluster size for an RK05 is 1. On an RP03, RECORD 4100% refers to device cluster number 4100 (which contains logical blocks 8200 and 8201) because its device cluster size is 2. In this case, the program accesses both blocks. The default buffer size for all disk units when open in nonfile structured mode is the device cluster size multiplied by 512 bytes. The following example reads the last two blocks of an RP03.

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

After the program opens the disk, the GET statement reads record 39999%, which contains the last two blocks of the disk.

The user can improve total throughput by specifying a large buffer size. This permits a single disk transfer to read a large quantity of data. To change the buffer size, the user includes the RECORDSIZE option in the OPEN statement. The RECORDSIZE specified should be an integral multiple of 512. For example, the following statement opens the disk on unit 1 for nonfile structured processing and sets the buffer size to 2048 bytes.

```
100   OPEN "DK1:" AS FILE 1%, RECORDSIZE 2048%
```

The system can access device cluster 0 only immediately following an OPEN statement. The GET or PUT statement that accesses device cluster 0 must either specify RECORD 0% or omit the RECORD option. Once the disk has been accessed, omitting the RECORD option or specifying RECORD 0% in a GET or PUT statement accesses the next sequential device cluster.

After the user program performs any I/O on the disk, the only way it can access device cluster 0 is by closing the disk and reopening it for nonfile structured access. The statement

```
100   OPEN "DK1:" AS FILE 1%:   GET #1%, RECORD 0%
```

reads the first block of an RK05.

CAUTION

On a RSTS/E file structured disk, logical block 0 contains the bootstrap. The remaining blocks, if any, in device cluster 0 contain no data. An attempt to write into device cluster 0 on a RSTS/E file structured disk destroys the bootstrap.

If the program attempts to read or write beyond the end of the disk, the END OF FILE ON DEVICE error occurs.

1.3.3 Privilege and Access

A disk that is being processed in nonfile structured mode need not be logically mounted. After a user inserts the disk into its drive, any user may read or write it. A nonprivileged user cannot read or write the disk in nonfile structured mode while another user is accessing it. Such an attempt results in a PROTECTION VIOLATION (ERR = 10) for the nonprivileged user. A privileged user can read the disk regardless of the number of users accessing it; but if a privileged user attempts to write on the disk while another user is accessing it, a PROTECTION VIOLATION error occurs.

If the disk is logically mounted, a privileged user gains only read access in nonfile structured processing. In this case, a nonprivileged user does not gain access to the disk.

By testing bits 9 and 10 of the BASIC-PLUS variable STATUS, the user program can determine what privileges it has. The STATUS variable is discussed in Section 12.3.5 of the *BASIC-PLUS Language Manual.*

1.4 FILE STRUCTURED DISK OPERATION

In file structured disk operation, the user organizes data into files. He uses the DSKINT option during system generation to set up a skeleton file structure on a RSTS/E disk. The user can create files with the OPEN and OPEN FOR OUTPUT commands, which are described in the *BASIC-PLUS Language Manual,* Section 9.2. Chapter 12 of that manual contains a complete discussion of Record I/O.

Disk files may be opened in one of several modes which are described in the following sections and summarized in Table 1-4. The general form of the OPEN statement with the MODE option is as follows:

```
    100   OPEN "PIG.DAT" AS FILE N%, MODE M%
```

N% is the internal I/O channel number and M% is the mode in which the file PIG.DAT is to be opened.

Table 1-4 MODE Specifications for Disk Files

MODE	Meaning
0%	Normal read/write (default)
1%	Update file
2%	Append to file
5%	Guarded update (4% + 1%)
8%	Special extend
16%	Create contiguous
4096%	Read normally regardless (privileged)
8192%	Read only
16384%	Write UFD or MFD (privileged)

1.4.1 Reading and Writing Disk Files – MODE 0%

By specifying MODE 0% or omitting the MODE option (i.e., using the system default), the user opens a disk file for normal reading and writing. In default mode, an OPEN FOR INPUT statement opens an existing file for read and write access (if the protection code of the file permits it). OPEN FOR OUTPUT deletes an existing file and creates a new file with the same name. An OPEN statement (without an INPUT or OUTPUT specification) attempts to perform an OPEN FOR INPUT operation. If this fails, the system creates a new file.

OPEN, OPEN FOR INPUT, and OPEN FOR OUTPUT control only the actions the system performs when it opens the disk file. These normal actions are described in the *BASIC-PLUS Language Manual.*

1.4.2 Updating Disk Files

In certain applications (for example, inventory updating) it may be necessary for multiple users to have read and write access to a single master file. In such cases, it is time consuming to continually close and reopen the file to obtain and relinquish write access. For this reason, RSTS/E provides an update option that allows multiple users to have write access to a file while guarding against simultaneous writing of the same data. The following sections describe the capabilities RSTS/E provides and those that are available through BASIC-PLUS.

1.4.2.1 RSTS/E File Updating Capabilities – In file updating operations, RSTS/E allows locks to be applied on blocks within a file. A single lock can apply to a single block or to a range of blocks. The blocks within the range of a single lock must be logically sequential; they need not be physically clustered. Because RSTS/E permits multiple locks at the same time on the same file, logically nonsequential blocks within a file can be updated in the same time period.

1.4.2.2 BASIC-PLUS File Update – MODE 1% – To open a file for update, the MODE 1% specification is used in the OPEN statement. For example:

```
100 OPEN 'MASTER.DAT' AS FILE 1%, MODE 1%
```

The statement opens MASTER.DAT for update on channel 1 and creates a 512-byte buffer in the user's job space. Because a file can not be simultaneously open in normal mode (MODE 0%) and in update mode, an attempt to perform an open in one mode when the file is currently open in the other mode generates the PROTECTION VIOLATION error (ERR=10). The same error occurs if the protection code of the file prohibits read access.

Although a file can not be open simultaneously in both normal and update mode, under a certain circumstance a program can still gain read access to the file. If the program has privilege, it can open the file with MODE 4096% described in Section 1.4.6. This mode allows normal read access regardless of whether the file is open for update.

After a program has opened a file for update, the system allows the program to access data simultaneously with other programs but enforces certain safeguards. When a program performs any read operation on the file, RSTS/E puts the block accessed in a locked state. An attempt by another program to access any data in that locked block results in the DISK BLOCK IS INTERLOCKED error (ERR=19). This error signals that the data required is being accessed on another channel in the current program or by another program and is perhaps being updated.

The program accessing the data makes the data available to another program by unlocking the block. Several ways, both implicit and explicit, exist for a program to unlock a locked block.

1. Perform any write operation on the file.
2. Execute the UNLOCK statement on the channel on which the file is open.
3. Read another block. (This action, however, locks the newly retrieved block.)
4. Execute a CLOSE statement on the file. [1]

Additionally, the system unlocks a block when the program encounters an error while accessing the file.

BASIC-PLUS allows a program to lock several logically consecutive blocks during a GET operation. The number of blocks is established by the RECORDSIZE option as shown in the following statement.

```
100   OPEN 'MASTER.DAT' AS FILE 1%,
      RECORDSIZE 1024%, MODE 1%
```

As a result of the 1024% in the RECORDSIZE option, BASIC-PLUS creates a 1024-byte buffer. A GET operation on channel 1, therefore, retrieves 2 blocks and puts both blocks together in the locked state. RSTS/E can allow up to 15 blocks to be locked in this manner but does not allow more than one lock. The same rules for a single locked block are applicable for the range of locked blocks.

1.4.2.3 BASIC-PLUS Guarded File Update – MODE 4% + 1% – MODE 5% in the OPEN statement enables the same update processing as MODE 1% but provides one additional processing feature. The program can write a block or range of blocks only after it has read and locked the data. Attempting to write data that is not currently locked results in the PROTECTION VIOLATION error (ERR=10). This feature prevents a program from updating data which it has not accessed. Note that 4% in the MODE option must be used only with 1% to gain special update. MODE 4% specified alone is equivalent to MODE 0%.

1.4.3 Appending Data to Disk Files – MODE 2%

To write data to a new block following the current end of file in a disk file, the user specifies MODE 2% in the OPEN statement. He should not use the OPEN FOR OUTPUT statement, as it deletes the existing file. The user should specify MODE 2% only with Record I/O files. For example:

Executing an END or CHAIN statement or running off the end of the program implicitly closes all files.

```
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. A PUT statement that the system later executes on the file need not specify a RECORD number. When the PUT statement does not include the RECORD option, the system writes the next sequential block.

The following sample program illustrates APPEND by showing its use in a specific instance. Each student enters his or her experimental data into a class data file. The complete class data file can then be input to another program to produce a class curve for the experiment.

```
100   DIM X(10%),X$(10%)
      \OPEN "SCIENC.EXP" AS FILE 1%, MODE 2%
      \IF (STATUS AND 1024%) THEN
              PRINT "WRITE ACCESS NOT GRANTED."
                    "TRY AGAIN IN A FEW MINUTES."
              \GOTO 800
400   FIELD #1%, 8%*I% AS B$, 8% AS X$(I%)
              FOR I%=1% TO 10%
500   PRINT "YOUR VALUES FOR X ARE";
      \MAT INPUT X
600   LSET X$(I%)=CVTF$(X(I%))
              FOR I%=1% TO 10%
700   PUT #1%
      \PRINT "THANK YOU"
800   CLOSE 1%
      \END
```

In certain applications, it is desirable to append records to a file on one channel and read the appended records on another channel. Because RSTS/E does not update the directory information on disk immediately after each append operation, a program must process the END OF FILE ON DEVICE error (ERR = 11) on the reading channel in a special manner. Upon encountering error number 11 when attempting to read appended records, the program should close the reading channel and reopen it. This action ensures that the latest directory information kept in memory is written to the disk. The appended records are then available on the reading channel.

1.4.4 Special Mode for Extending Files – MODE 8%

To force RSTS/E to update a file's size data on the disk during extend operations, the MODE 8% option is used in the OPEN, OPEN FOR INPUT, or OPEN FOR OUTPUT statement. In normal processing, RSTS/E maintains a file's size data in memory. This size is not updated on disk until a new cluster is allocated to the file. By specifying MODE 8%, the program forces RSTS/E to update the on-disk file size for every block added to the file.

The following sample statement shows the MODE 8% option.

```
10      OPEN 'DATA' AS FILE 1%, MODE 8% + N%
```

The system creates the file if it does not exist. The value N% can be any other disk MODE option.

Extending a disk file using MODE 8% increases the processing overhead because the system must access the disk more times for every block added. The extra overhead is warranted for applications where a file's size must be correctly preserved in the event of a system crash or power failure.

1.4.5 Creating a Contiguous File – MODE 16%

MODE 16% can be specified with the FILESIZE option in the OPEN FOR OUTPUT statement to create a contiguous file on disk. Contiguous means that the clusters allocated to the file are logically adjacent. The following sample statement shows the procedure to create a contiguous file.

```
10      OPEN 'DATA.1' FOR OUTPUT AS FILE 1%,
        FILESIZE 12%, MODE 16% :
```

Other options can be used with MODE 16% to specify the buffer size (RECORDSIZE) and the file cluster size (CLUSTERSIZE). The FILESIZE option is necessary with MODE 16%. It pre-extends the file to its maximum length, thus telling the system how much contiguous space is required. If sufficient contiguous space is not available, the system generates the NO ROOM FOR USER ON DEVICE error (ERR=4).

Processing a contiguous file reduces overhead greatly because directory accesses and movement of read/write heads is minimized. Files for run-time systems and swapping must be contiguous because the monitor accesses these files independently of the normal file processor. A contiguous file, however, cannot be extended. An attempt to extend a contiguous file generates the PROTECTION VIOLATION error (ERR=10).

1.4.6 Reading a File Normally Regardless – MODE 4096%

In certain applications it is advantageous to be able to read a data file regardless of what other processing is transpiring. Under normal circumstances, the system prohibits opening a file normally while the file is currently open for update (MODE 1% or MODE 4%+1%). With MODE 4096%, a privileged program can open a file for read access regardless of whether the file is being updated.

The following sample statement shows the procedure.

```
10      OPEN 'DATA.2' FOR INPUT AS FILE 1%,
         RECORDSIZE R%, MODE 4096%
```

Write operations are not permitted. If a write operation is attempted, the system generates the PROTECTION VIOLATION error (ERR=10). If the file is simultaneously open for update, the normal DISK BLOCK IS INTERLOCKED error is not generated when the program reads a block being updated.

MODE 4096% requires privileges because of the danger involved in reading data which is perhaps changing.

1.4.7 Read Only Access to a File – MODE 8192%

Certain applications require simple read access to a data file and do not want to preclude write access for other applications. Under normal circumstances, an OPEN FOR INPUT statement for a disk file possibly gains write access on the I/O channel involved. To bypass the mechanism which grants write access, the MODE 8192% option is specified in the OPEN FOR INPUT statement.

The following statement shows the procedure

```
10      OPEN 'DATA.3' FOR INPUT AS FILE 1%,
         RECORDSIZE R%, MODE 8192%
```

After execution of the above statement, the program has only read access to the file DATA.3. If the file is currently open for update, however, the system generates the normal PROTECTION VIOLATION error (ERR=10).

1.4.8 Writing the UFD or MFD – MODE 16384%

A privileged user can write into a UFD or MFD by specifying MODE 16384% in the OPEN statement. For example, the following statement allows a privileged user to read and write into the UFD of account [5,10]:

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

To write the MFD, the user specifies [1,1] in the file name. An OPEN FOR OUTPUT statement is invalid for a UFD or MFD. Without MODE 16384%, only read access is allowed.

1.4.9 Simultaneous Disk Access

RSTS/E permits several users to read from the same file simultaneously, but only one at a time to write into a single file. Without this limitation, two users could try simultaneously to write the same record of the file, resulting in a loss of data. To avoid this conflict, the system permits only one user at a time to have write access to any file (unless the file is open in update mode). If a second user attempts to write into the file, a PROTECTION VIOLATION error results. Thus, a user may fail to obtain write privileges to a file that is not write-protected against him. If this failure occurs, the second user must close the file and reopen it after the first user has closed it.

The system does not permit a file to be open simultaneously in update mode and in normal mode.[1] Attempting to do so results in a PROTECTION VIOLATION (ERR=10). A file may be open in update mode by multiple users.

By checking bits 9 and 10 of the STATUS variable immediately after the OPEN statement, a program can ascertain whether the current job has read and write access to a file.

The STATUS variable is described in Section 12.3.5 of the *BASIC-PLUS Language Manual*. The example given in Section 1.4.3 performs such a check.

1.4.10 Disk Optimization

Whenever a user opens a file on the public structure, the system searches the directories of all public disks. This search determines whether the file exists. The user avoids the overhead of searching multiple directories by putting the file on a private disk.

Dedicating a private disk to a large production file minimizes overhead to access data and ensures an efficient directory organization. If the user finds this impractical and must store more than one such file on one private disk, he should dedicate an entire account to each file. This arrangement reduces directory search overhead.

However, if the user must save more than one file under an account, he should create the more frequently accessed ones first to ensure better directory organization. If he cannot do this, the system manager can optimally reorder the file directory with the REORDR system utility, which is described in the *RSTS/E System Manager's Guide*, Section 7.6. Thus, files on an account can be ordered in either forward or reverse direction, by either date and time of creation or date of last access.

When creating a large file, the user should specify a large file cluster size to increase efficiency. This reduces the number of UFD blocks required to describe the file. Throughput improves because the system can read or write multiple blocks in a single transfer operation. In addition, the user can pre-extend a disk file to its maximum length when he creates it and can specify that contiguous space be used. Pre-extension reduces directory fragmentation. Contiguous space reduces window turning.

[1] A file may, however, be open simultaneously in update mode and read normally regardless mode as described in Section 1.4.6.

If a program requires simultaneous access to more than one data file, the optimal organization places each file on a different private disk. Overhead increases if the files reside on the same disk because the disk head must move whenever the program accesses a different file. Thus, a large percentage of execution time is spent in moving the disk head back and forth.

Different accounts should be used to store files. Certain accounts can be dedicated to files that are created once and remain fairly static. Other accounts can be reserved for transient files. Thus, the number of poorly ordered accounts is minimized. To further optimize the structure, minimize the number of files in one account. For example, it is better to have 30 files each in 10 accounts than to have 300 files in one account.

1.4.11 Partial Block Operations on Disk

The RECORDSIZE option in the OPEN statement on a disk file can specify a value which is not a multiple of 512 bytes. Care, however, must be exercised in using the GET statement. For example, the following statement creates a 520 byte buffer on channel 1.

```
10      OPEN 'MASTER.DAT' AS FILE 1%, RECORDSIZE 520%
```

A subsequent GET operation on channel 1% fills the buffer to the requested number of bytes. The disk software then skips the rest of the last disk block read and positions itself to access the next block. To fill the 520-byte buffer, the software reads the current block (for 512 bytes), reads 8 bytes of the next block, and positions itself to access the following block.

For GET operations, any value for RECORD may be used. For example, with a buffer of 520 bytes, RECORD 1 retrieves the first block and 8 bytes of the second block. RECORD 2 in the GET statement will retrieve the entire contents of the second block plus 8 bytes of the third block. The inter-block positioning is not maintained.

For PUT operations, only blocks with a size which is a multiple of 512 bytes are written.

1.5 FLOPPY DISKS

The RX11/RX01 Floppy Disk can be used only as a non-file structured device.[1] 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 summarized in Table 1-5.

[1]The FLINT system program is available to allow users to transfer specially formatted data to the RSTS/E environment.

Table 1-5 MODE Specifications for Floppy Disk

MODE	Meaning
0%	Read and write in block mode (default)
16384%	Read and write in sector mode

The following sections describe the MODE specifications.

1.5.1 Sector Mode – MODE 16384%

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% and 1%) is included in the RECORD option (e.g., RECORD N%+32767%+1%), sectors are interleaved according to the algorithm discussed in Section 1.5.2.

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 1.5.2) and 16384% to write a Deleted Data Mark with each of the records (see Section 1.5.3). C% must be a multiple of 128.

1.5.2 Block Mode – MODE 0%

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 1.5.3). 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.

1.5.3 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 record size 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.

1.5.4 Partial Block Operations on Floppy Disks

The RECORDSIZE option in the OPEN statement on a floppy disk can specify a value which is not a multiple of the default buffer size (512 bytes in block mode and 128 bytes in sector mode). Care, however, must be exercised in using the GET and PUT statements.

For GET operations with other than the default buffer size (or a multiple of the default), the software retrieves the required number of bytes and positions itself to the next boundary. In block mode, this boundary is the next block (sector number times 4); in sector mode, this boundary is the next sector. Thus, for a buffer size of 520 bytes, a GET statement in block mode returns in the buffer the current sector, the next 3 sectors and the first 8 bytes of the fourth sector. The software then skips the rest of the fourth sector and all of the fifth, sixth and seventh sectors to position itself at the beginning of the next block boundary for the next GET operation. A GET statement in sector mode returns the required number of bytes and skips the rest of the partial sector to position itself at the beginning of the next sector boundary.

Any value (with the limits) may be used in the RECORD option with the GET statement. Thus, with a buffer size greater than 512 bytes, record values may be overlapped to recover skipped data.

For the PUT operation, with other than the default buffer size (or a multiple of the default), the software performs the same skipping and positioning as with the GET statement. The software writes zero-valued bytes in the skipped data. If the COUNT option is included in the PUT statement, the software writes the specified number of bytes from the buffer and writes zero-valued bytes for the rest of the buffer and for the skipped data.

1.6 THE NULL DEVICE – NL:

The null device exists on all RSTS/E systems as a debugging aid. It provides a means for a program to check out all I/O routines without reference to an actual device. A read access for the null device returns the END OF FILE ON DEVICE error (ERR=11) and a write access simply returns control.

The null device can be used to dynamically allocate buffer space in memory. It has a default buffer size of 2 which is adequate for performing alternate buffer I/O operations with data on another channel. The buffer size can be established with the RECORDSIZE option in the OPEN statement.

The null device is sharable by all users on the system and no user can assign it.

CHAPTER 2
MAGTAPE

Magtape I/O is processed under RSTS/E 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 and ANSI magtape file labels.)

NOTE

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 with the MODE specification in the OPEN statement. MODE establishes how the system searches for a file with the OPEN, OPEN FOR INPUT, and OPEN FOR OUTPUT statements described in Section 9.2 of the *BASIC-PLUS Language Manual.* It also controls the system action when the file is subsequently closed.

RSTS/E normally writes tape records 512 bytes long. Table 2-1 lists standard system defaults for magtape.

Table 2-1 System Parity and Density Defaults[1] for Magtape

TU10 7-Track	**TU10 TS03 9-Track**	**TU16 TU45 9-Track**
800 bpi ODD parity DUMP mode	800 bpi ODD parity	800 bpi ODD parity

A user program can override the system defaults by system or CCL commands (ASSIGN or MOUNT) and can override the assigned defaults by program operations (MAGTAPE function for both non-file structured and file structured and the MODE option for non-file structured) described in this chapter.

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%
```

[1] An optional patch may be installed to change system magtape parity and density defaults. Consult the *RSTS/E Release Notes* for details.

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 in Table 2-2 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.

Table 2-2 Magtape OPEN FOR INPUT MODE Values

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.

Under DOS file structured operations, the system reads magtape records into a 512-byte buffer. In certain cases, however, the user may need to process records larger than 512 bytes. With the RECORDSIZE option, the user program can allocate more buffer space than the default provides. The form of the statement is:

```
    100 OPEN "MT0:FIDO" FOR INPUT AS FILE N%, RECORDSIZE M%
```

N% is the internal I/O channel on which the file is open and M% is the desired record length. The system rounds M% down to an even number if M% is odd. This statement opens the file FIDO on magtape unit 0 for input and allocates M% bytes of buffer space for data transfer operations.

For ANSI file structured operations, the system reads the block length from the header 2 (HDR2) label when it opens the file. The buffer is created at the size given by the block length. If the block length is odd, however, the system rounds the value down to make the buffer size an even number of bytes.

2.1.1 Searching for a Label

Omitting the MODE specification or using a MODE 0% specification reads the record at the current position of the tape. The system expects the label format to be the system-wide default unless the format was changed when the unit was assigned to the job. If the label format differs, the system generates the BAD DIRECTORY FOR DEVICE error (ERR = 1). No match causes the system to rewind the tape and check successive label records until the label 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 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%.

Including MODE value 64% with any of the above modes 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 either specified by the system manager when he starts time sharing operations or specified by the user when he reserves the unit to the current job (ASSIGN, MOUNT or the assign SYS function call). The MODE value overrides any current defaults for labeling.

Including MODE 24576% (16384% + 8192%) 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 format in which the tape is written differs from that used in the search, the system generates the BAD DIRECTORY FOR DEVICE error (ERR = 1). 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 default label format.

2.1.4 The GET Statement

The GET statement reads a single record into the I/O buffer from a magtape file that is OPEN FOR INPUT. The form of the GET statement for magtape is:

```
100 GET #N%
```

N% is the channel on which the device is open. This statement reads the next sequential record in the file. For DOS format tapes, the buffer is 512 bytes long unless the user specifies a larger buffer with the RECORDSIZE option when he opens the file. For ANSI format tapes, the buffer size is the block length read from the HDR2 label. Magtape hardware allows only sequential access. Therefore, the RECORD option cannot be used. The number of bytes actually read is available in the RECOUNT variable. To associate string names with all or part of the data in the I/O buffer, use a FIELD statement, described in Section 12.4.2 of the *BASIC-PLUS Language Manual*. Attempting to read beyond the end of file results in END OF FILE ON DEVICE (ERR = 11).

2.1.5 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 associates it with channel 3%. MODE 24772% is 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 meanings shown in Table 2-3 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.

Table 2-3 Magtape OPEN FOR OUTPUT MODE Values

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 Label

Omitting the MODE specification or using a MODE 0% specification reads the tape at its current position. The system expects the label format to be the system wide default unless the format was changed when the unit was assigned to the job. If the label format differs, the system generates the BAD DIRECTORY FOR DEVICE error (ERR = 1). 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 Label

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.

NOTE

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 unless 64% was included in the MODE value.

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.

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 tue 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 default format either specified by the system manager when he starts time-sharing or specified by the user when he reserves the unit to the current job (ASSIGN or MOUNT). The MODE value overrides any current defaults for labeling.

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 format in which the tape is written differs from that used in the search, the system generates the BAD DIRECTORY FOR DEVICE error (ERR = 1). 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 values 16384% or 24576% in the OPEN FOR OUTPUT statement assumes record labels are in the system-wide or job-related default format.

2.2.6 Processing DOS Magtape Files

If the tape being processed is in DOS format, the RECORDSIZE option in the OPEN FOR OUTPUT statement designates the block length. Omitting the RECORDSIZE option from the OPEN FOR OUTPUT statement is equivalent to specifying RECORDSIZE 0. That is, BASIC-PLUS creates a 512-byte buffer, the default for DOS magtape processing. PUT statements write blocks on tape equal to the buffer size – 512 bytes.

To write blocks larger than 512 bytes, simply specify an even value equal to or greater than 512 in the RECORDSIZE option. If the value is odd, BASIC-PLUS rounds down the buffer size to make it even.

To write blocks smaller than 512 bytes, the program can create a buffer smaller than 512 bytes. To create a buffer smaller than 512 bytes, specify 32767%+1% plus an even value equal to or greater than 14 in the RECORDSIZE option. Fourteen bytes is the minimum length the magtape hardware can handle. The 32767%+1% value sets the sign bit and tells BASIC-PLUS to use the value specified rather than the default value of 512. If the sign bit is not set, a 512-byte buffer is created. If the value given is odd (and the sign bit is set), the buffer size is rounded down to make it even. PUT statements without the COUNT option write blocks on tape equal to the buffer size. The COUNT option can be used to write tape blocks smaller than the buffer size but not less than the minimum of 14 bytes.

2.2.7 Processing ANSI Magtape Files

If the tape being processed is in ANSI format, the CLUSTERSIZE and FILESIZE options in the OPEN FOR OUTPUT statement designate record format and length, file characteristics, and block length. The FILESIZE and CLUSTERSIZE options have effect only when the tape being processed is in ANSI format. The general form of the statement with options is:

```
10 OPEN 'MT0:ABC' FOR OUTPUT AS FILE N%,
     CLUSTERSIZE Q%, FILESIZE F%, MODE 24576% + M%
```

The options must be specified in the exact order shown; otherwise, the system generates the MODIFIER ERROR. To apply the system default for any option, omit that specification from its place in the statement.

The system associates magtape unit 0 with the channel designated by N%. The system searches for file ABC according to the value specified by M% in the MODE option described above. The value 24576% in the MODE option ensures that ANSI format processing is done because any system or device defaults are overridden by the value in the MODE option. For the search to be successful, the file name ABC must match the file identifier in the header 1 label (HDR1) on the tape.

The value Q% in the CLUSTERSIZE option designates the record length, record format and characteristics of the file created. The value given causes the system to write the appropriate data in the label fields of the header and end of file records on tape. Table 2-4 shows the related label data for values of Q%. The value specified with CLUSTERSIZE is the sum of values chosen from Table 2-4.

Table 2-4 ANSI Magtape CLUSTERSIZE Values

Label Field Name	CLUSTERSIZE Value	Label Result
Record Format	0% 16384% 32767%+1% -16384%	U (undefined) F (fixed length) D (variable length) S (spanned)[1]
Record Length (in bytes)	Between 0% and 4095%	For U, always 0%. For F, default is 512%. For D, value gives maximum record length. For S, value is unused.[1]

System Dependent (File Characteristics)	0%
	4096%
	8192%
		M = carriage control embedded
		A = FORTRAN carriage control
		(space) = Line feed precedes and carriage return follows each record

[1]RSTS/E does not support ANSI format S records.

If the CLUSTERSIZE option is omitted from the OPEN FOR OUTPUT statement, the system applies CLUSTERSIZE 0%. That is, a file is created with undefined (U) record format and embedded carriage control with record length 0%.

The record length specified in the CLUSTERSIZE option is the value which the system writes in character positions 11 through 15 of the header 2 (HDR2) label record. For fixed length records, this value should equal the number of bytes the program uses in the FIELD statement to subdivide the I/O buffer. The subdivisions created to load records into the I/O buffer will then equal the record length on tape label. For variable length records, this value should be the maximum length of a record.

The value P% in the FILESIZE option designates the block length for the file. The system writes this value in character positions 6 through 10 of the header 2 (HDR2) label when it opens the file. If the FILESIZE option is omitted (equivalent to specifying FILESIZE 0%) from the OPEN FOR OUTPUT statement, the system sets the block length to 512 bytes. The value specified in the FILESIZE option must be between 14 (the minimum allowed by the hardware) and 4095.

In ANSI processing, the system uses the block length from the HDR2 label to create the magtape I/O buffer. This action enables the program to write blocks of records on tape equal in size to the I/O buffer. The block length in the FILESIZE option should correspond to the total size of the I/O buffer defined by the FIELD statement.

In ANSI processing, therefore, the FILESIZE option enables the program to create an I/O buffer other than 512 bytes. The block length on the tape should be an even number. If the number is odd, the system rounds it down 1 byte to make the I/O buffer an even number of bytes.

NOTE

The action of the FILESIZE option in ANSI processing appears similar to the RECORDSIZE option. However, if the RECORDSIZE option is used in ANSI processing and the value it specifies is larger than the block length in the HDR2 label, the system establishes the I/O buffer at the size given in the RECORDSIZE option. No conceivable advantage is gained from using a buffer size larger than the block length. It is therefore recommended that the RECORDSIZE option not be specified in ANSI processing.

The PUT statement without the COUNT option in ANSI processing writes blocks equal to the buffer length. By creating a buffer larger than 512 bytes, the program can write blocks to tape equal to the buffer length and greater than the default of 512 bytes.

To write blocks shorter than 512 bytes, the program simply specifies in the FILESIZE option the desired block length (an even number between 14 and 512). The system writes the block length properly to the HDR2 label and establishes the I/O buffer equal to the block length. (This action differs from DOS processing where the block length is not available and the system sets the buffer length to 512 bytes.) To write the block at the correct length (that is, the length which matches what is written on the HDR2 label), the program uses the PUT statement without the COUNT option. The system writes a block at the size of the I/O buffer (which matches the block length in the HDR2 label).

2.2.8 The PUT Statement and Processing End of Tape

The PUT statement writes the contents of the I/O buffer for the specified I/O channel to the next sequential record of the file. The form of the PUT statement is:

```
    100   PUT #N%
```

N% specifies the internal channel on which the file is open. PUT writes a single record to a magtape file.

If RSTS/E encounters the reflective end-of-tape marker while writing to tape, the system writes the entire record and returns the error NO ROOM FOR USER ON DEVICE (ERR = 4). The error condition does not harm the data. GET statements (when the file is later opened for input) access data at and beyond the reflective marker without error. RSTS/E provides two recovery procedures:

1. The user can close the file as soon as the error occurs, then create another file on another tape for the remainder of the data.
2. If the user wants one file to contain all the data, he may include in the program a subroutine that writes a logical end-of-tape mark at the end of the previous file. The user may then write the file that generates the error condition to another tape. The steps in this subroutine are as follows:

   a. Backspace with the MAGTAPE function using the maximum parameter 32767% (see Section 2.8.5). This procedure moves the tape to an end-of-file (EOF) or beginning-of-tape (BOT) mark.
   b. If no error or the NO ROOM FOR USER ON DEVICE error occurs during the backspace, check the tape status function (see Section 2.8.7) to ascertain whether the tape is at EOF or BOT. (If any other error occurs, the data may be corrupt.)
   c. If the tape is at BOT, the file will not fit on the tape. Write three EOF marks (see Section 2.8.2) to zero the tape, then try a longer tape. Encountering BOT should occur only on DOS tapes. ANSI magtape files contain an EOF character in the label record; thus, the system should find an EOF before encountering BOT.
   d. If the tape is at an EOF mark and is in DOS format, write three EOF marks. On an ANSI tape, backspace to the next tape mark, then write three EOF marks.

2.2.9 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 specified 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.

If the system cannot open the file for input, it performs an OPEN FOR OUTPUT operation using the MODE M% specification. OPEN FOR INPUT or OPEN FOR OUTPUT should be used instead of OPEN with magtape. OPEN FOR INPUT and OPEN FOR OUTPUT enable the program to determine immediately which operation is needed.

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 or TS03 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 or TU45 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+S

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

S (stay) is:

0	= MODE value does not stay
8192	= MODE value stays (is retained) after CLOSE

If mode is not specified in the OPEN statement, the tape is processed using the system parity and density defaults.[1]

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.[1] If any other values are used, the system recognizes only the parity specification.

MODE for a 9-track TU16 or TU45 drive can indicate 1600 BPI phase encoded operation. Parity is always odd for phase encoded operation.

By adding 8192 to the MODE value, the associated parity and density setting remains in effect for the job even after the channel has been closed.

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%
```

[1]An optional patch may be installed to change system magtape parity and density defaults. Consult the *RSTS/E Release Notes* for details.

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.

To read and write records larger than 512 bytes, include the RECORDSIZE option in the OPEN statement. For example, the statement:

```
100   OPEN "MT0:" AS FILE 1%, RECORDSIZE 1000%, MODE 12%
```

associates the tape on magtape unit 0 with channel 1. The RECORDSIZE option creates a buffer of 1000 bytes. If insufficient memory is available, the MAXIMUM MEMORY EXCEEDED error is generated. The user must either reduce the size of the program or increase maximum size to which the job can grow. The buffer length must be an even number greater than 512. If the number given is odd, the system rounds it down 1 byte to make it even. If the number is less than 512, the system uses the default buffer length of 512 regardless.

Subsequent GET and PUT operations on channel 1 read and write records 1000 bytes long. Attempting to read a record longer than the buffer generates the MAGTAPE RECORD LENGTH ERROR (ERR = 40). The RECOUNT variable contains the number of bytes read.

To write records shorter than the buffer size, open the device normally and specify the COUNT option in the PUT statement. The statement:

```
205   PUT #1%, COUNT 76%
```

writes a 76 byte record. When COUNT is not used, PUT writes an entire buffer, regardless of whether the buffer contains data. A record must be at least 14 bytes (a hardware limitation) and no larger than the I/O buffer. If a record smaller than the buffer size is read, the RECOUNT system variable contains the number of bytes read. Every input operation on any channel (including channel 0) sets RECOUNT. Thus, the user program should test it immediately after each GET statement.

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. Unless otherwise stated, a MAGTAPE function works only in non-file structured operations. 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.

Table 2-5 summarizes the MAGTAPE function codes.

Table 2-5 MAGTAPE Function Summary

Action	Function Code	Parameter	Value Returned
Rewind and off-line	1	unused	0
Write end-of-file	2	unused	0
Rewind	3	unused	0
Skip record	4	records to skip	# records not skipped
Backspace over record	5	records to back-space	# records not back-spaced
Set density and parity	6	E+D*4+P+S	0
Tape status function	7	unused	status
File characteristics	8	unused	file characteristics
Rewind on CLOSE	9	unused	0

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.

[1]The system counts the EOF record as a record skipped.

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).[1] 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+S
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

S = Stay

0	= MODE value does not stay
8192	= MODE value stays (is retained) after CLOSE

[1]The system counts the EOF record as a record actually backspaced.

A magtape drive is set to the system default for density and parity unless the default is changed when the unit is assigned (with an ASSIGN statement) or when the unit is 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 I% to 0 to indicate successful completion. This function works for both non-file structured and file structured operations.

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 without a MODE option, associating any channel number with magtape unit 0, opens it with that new density/parity setting automatically. A DEASSIGN statement to a previously assigned unit reverts the density/parity setting for the magtape unit to the system default value. Specifying another parameter value also changes the density and parity setting. The setting remains if ownership of the unit is passed to another job.

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 operation 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 2-6. The example shown below obtains the status of the magtape opened on internal channel number 2:

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

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 2-6 as follows:

Table 2-6 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 at 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 or TU45
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

```
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 (1% 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 GOTO 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 2-7.

Non-zero integers are returned for ANSI files; zero is returned for DOS 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 (16384% + 64% + 16%) 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 2-7, as follows:

Table 2-7 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

[1]ANSI format S is not supported by RSTS/E systems.

```
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.

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, instructs the system not to rewind on closing the file. Once the Rewind on CLOSE function is executed, it cannot be cancelled.

2.9 MAGTAPE PROGRAMMING EXAMPLES

2.9.1 Writing a Magtape File

The BASIC-PLUS program which follows opens an existing magtape file for output and appends data to the file.

```
100   M%=16384%+128%+64%+32%
      \OPEN "MT0:RECORD.FIL" FOR OUTPUT AS FILE 1%, MODE M%
      \FIELD #1%,2% AS S$, 8% AS M$, 2% AS Y$, 8% AS C$, 2% AS D$
      \INPUT "HOW MANY RECORDS TO ENTER";A%
400   FOR I%=1% TO A%
              \INPUT "RECORD ";S%
              \INPUT K$
              \INPUT Y%
              \INPUT L$
              \INPUT D%
500           LSET S$=CVT%$(S%)
              \LSET Y$=CVT%$(Y%)
              \LSET D$=CVT%$(D%)
              \LSET M$=K$
              \LSET C$=L$
              \PUT #1%, COUNT 22%
      \NEXT I%
      \CLOSE 1%
3000  END
```

The program opens the file RECORD.FIL, which is on a DOS tape (MODE 16384%), for append (MODE 128%). The system rewinds the tape before it searches for the file and when it executes a CLOSE statement on the file (MODE 32%+64%). After the user types in each record, the program converts the data as necessary and writes the record to the file. After all records have been written, the program closes the file and ends.

2.9.2 Reading a Magtape File

The following program opens a magtape file for input and reads records from the file. It assumes a file in which records are identifiable by an integer key.

```
150    M%=16384%+64%+32%
       \OPEN "MTO:RECORD.FIL" FOR INPUT AS FILE 1%, MODE M%
200    INPUT "HOW MANY RECORDS"; F%
210    FOR I%=1% TO F%
       \N%=0%
       \INPUT "RECORD TO FIND";J%
300    GET #1%
       \FIELD #1%, 2% AS S$, 8% AS M$, 2% AS Y$, 8% AS C$, 2% AS D$
500    N%=N%+1%
       \S%=CVT$%(S$)
       \GOTO 300 IF J%<>S%
625    Y%=CVT$%(Y$)
       \D%=CVT$%(D$)
750    PRINT S%
       \PRINT M$
       \PRINT Y%
       \PRINT C$
       \PRINT D%
       \T%=MAGTAPE(5%,N%,1%)
       \NEXT I%
       \CLOSE 1%
2000   END
```

The program opens the magtape file RECORD.FIL on I/O channel 1 with read access only. The tape is in DOS format and is rewound both before the system searches for the file and when the system closes the file (MODE 16384% + 32% + 64%). The program causes a search for the record the user has specified and converts the data in the record to a recognizable form before printing it. The MAGTAPE function backspaces the tape to the beginning of the file following each record retrieval so that the user may request records in any order. Finally, the program closes the file and ends.

2.9.3 Reading a Tape Nonfile Structured

The following program reads a DOS magtape label record.

```
100    DEF FNZ$(Z$)=RAD$(SWAP%(CVT$%(Z$)))
110    INPUT "WHICH DRIVE";M$
       \OPEN M$ AS FILE 1%
200    FIELD #1%, 2% AS F$, 2% AS N$, 2% AS X$, 1% AS P$, 1% AS J$,
       1% AS C$, 1% AS U$, 2% AS D$, 2% AS U1$
       \GET #1%
250    F1$=FNZ$(F$)+FNZ$(N$)+"."+FNZ$(X$)
300    P%=ASCII(P$)
       \J%=ASCII(J$)
       \C%=ASCII(C$)
400    D%=SWAP%(CVT$%(D$))
       \Y$=DATE$(D%)
500    PRINT F1$,P%,J%,C%,Y$
600    CLOSE 1%
32767 END
```

The program opens the tape for non-file structured processing on I/O channel 1. No MODE specification is necessary because the tape is 9-track, 800 bpi, odd parity. After reading the 14-byte label record, the program converts the filename (bytes 0-5) from Radix-50 notation to the ASCII character string F1$. The program then converts the project-programmer number and protection code (P$, J$, and C$) to integer format. It next changes the creation date of the file (D$) to PDP-11 internal form and uses the DATE$ function to obtain the creation date in DD-MMM-YY format. Finally, the program prints all the label information and ends.

2.10 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, write lock error and write beyond EOT error.

2.10.1 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 (ERR=13) 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.

2.10.2 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, USER DATA ERROR ON DEVICE error (ERR=13) 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.

2.10.3 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.

2.10.4 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 (ERR=14).

2.10.5 Writing Beyond EOT Error

Attempting to write a record beyond the end-of-tape reflective marker writes the entire record but returns the error NO ROOM FOR USER ON DEVICE (ERR=4). This error condition is a warning to the user program and does not harm the data. The program can recover in one of two ways as described in Section 2.2.8.

2.11 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 Resultant Action	LP11 Resultant Action
CHR$(7)	BELL (A 2-second audible tone)	None
CHR$(8)	BS – backspace 1. Prints line 2. Returns carriage 3. Spaces to position immediately before previous position on line	BS – backspace 1. Prints line 2. Returns carriage 3. Spaces to position immediately before previous position on line
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. Prints line 2. Returns carriage 3. Advances paper one line	PF – Paper Feed 1. Prints line 2. Returns carriage 3. Advances paper one line
CHR$(11)	VT – Vertical Tab 1. Advances paper to the next hole position in Channel 5	VT – Vertical Tab None
CHR$(12)	FF – Form Feed 1. Prints line 2. Returns carriage 3. Advances paper to the next hole position in Channel 7 (see Section 3.2)	FF – Form Feed 1. Prints line 2. Returns carriage 3. Advances paper to the third line of the next form (hardware top of form, see Section 3.2)

(continued on next page)

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

Character	LS11 Resultant Action	LP11 Resultant Action
CHR$(13)	CR – Carriage Return 1. Prints line 2. Returns carriage 3. No line feed (may be used for overprint)	CR – Carriage Return 1. Prints line 2. Returns carriage 3. No line feed (may be used for overprint)
CHR$(14)	ELONG –Elongated Character 1. Doubles the horizontal printing axis	ELONG – Elongated Character None
CHR$(17)	SEL – Select 1. Allows the software to put the printer on-line	SEL – Select None
CHR$(19)	DSEL – Deselect 1. Allows the software to put the printer off-line	DSEL – Deselect None
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 None	DEL – Delete None

3.2 LINE PRINTER CONTROL VIA THE MODE OPTION

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 MODE values shown in Table 3-2 generate the action described at the line printer unit.

The following sections describe the various uses of the MODE option.

3.2.1 Handling Nonstandard Forms – MODE 512% + N%

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 512%+30%
```

The statement sets the form length to 30 lines per page. 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 0 to one less than the length specified. Thus, in this example, lines are numbered from 0 to 29.

Table 3-2 Line Printer OPEN MODE Values

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 >128.
1024%	Translate lower case characters to upper case characters. Applies only to upper and lower case line printers.
2048%	Skip 6 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.)
8192%	Suppress form feed on CLOSE.

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 128 from the decimal value of the special character the program sends to the printer. For example,

```
PRINT #1, CHR$(128%+19%);
```

This statement causes the system to evaluate the difference between 147 and 128. If the difference is greater than the page length 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.

The system treats characters whose values lie between 0 and 127 as the standard ASCII equivalents. If MODE 512% is not specified in the OPEN statement, characters whose values lie in the range 128% to 255% are treated as (value - 128%).

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

3.2.2 Hardware Form Feed – MODE 4096%

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 form feed 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.

NOTE

If both 4096% and 512% values are included in the MODE option, a form feed (FF) character sent to the line printer remains untranslated. The form feed causes the top of hardware form. This action results in unpredictable output because the line counting done by the MODE 512% processing does not take into account the movement of the paper to the top of hardware form.

3.2.3 Translating Numeric 0 To Literal O – MODE 128%

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 512%+128%+66%
```

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.

3.2.4 Truncating Long Lines – MODE 256%

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 512%+256%+128%+66%
```

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.

3.2.5 Translating Lower Case to Upper Case – MODE 1024%

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 1024%+512%+256%+128%+66%
```

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$(96) and CHR$(126) to their equivalents between CHR$(64) and CHR$(94). This feature is always set for an upper case only printer.

3.2.6 Skipping Lines at Perforation – MODE 2048%

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

```
10   OPEN "LP0:" 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 fourth 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.

3.2.7 Suppressing Form Feed on CLOSE – MODE 8192%

For certain applications, it is necessary to maintain the current print position on the line printer during a CLOSE operation. Normally, the system automatically generates a form feed on either an implicit CLOSE (as in a CHAIN operation) or an explicit CLOSE. By specifying MODE 8192% in the OPEN statement, the program tells RSTS/E not to generate the form feed when the CLOSE operation is performed on the channel open for the line printer.

The following statement shows the procedure.

```
    10      OPEN 'LPO:' AS FILE 1%, MODE 8192% + N%
```

The value N% can be any other combination of MODE values valid for line printer operation.

3.3 LINE PRINTER CONTROL VIA RECORD OPTION

The RECORD option in a PUT or PRINT statement modifies the operation of the line printer and enables discrete control of individual output steps. Table 3-3 lists the values allowed in the option.

Table 3-3 Line Printer RECORD Values

Value	Meaning
2%	Print over perforation (disables MODE 2048% for this output step)
4%	Do not return control to the program until output is complete or until an error is encountered
8%	Clear pending output buffers before buffering characters for the request

The general format of the RECORD option for line printer operation is shown below.

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

or

```
    10      PRINT #N%, RECORD R%, A$
```

The following sections describe the RECORD values.

3.3.1 Print Over Perforations – RECORD 2%

By specifying RECORD 2% in the PUT or PRINT statement, the program can temporarily override the effect of MODE 2048% on an output form. For example, an application program, which is usually skipping six lines at the bottom of forms, might need to print an identification or special page requiring all lines on the page. RECORD 2% allows the program to print in the lines normally skipped.

3.3.2 Delay Return Until Output Complete – RECORD 4%

For line printer output, the system transfers data from program storage to the device by intermediate storage called small buffers. This intermediate buffering allows the faster computational process to continue unhindered by the slower output action at the line printer. For each output request, the system transfers the data to small buffers. At the same time, at its own speed, the line printer software extracts the data from the small buffers and performs the output to the device.

Normally, completion of an output request occurs when the data is buffered. Upon completion of the buffering, the system returns control to the program at the next statement. The characters to be printed reside in the buffers as the physical print operation proceeds. If the program finishes its output routine and an error occurs at the device before the data is actually printed, recovery can be difficult under programmed control.

The RECORD 4% option on an output request tells the system not to return control until the data is actually printed. This mechanism allows a program greater control over error recovery. To use this mechanism, it is suggested that programs print a NUL character with the RECORD 4% option. The following statement shows the procedure.

```
    10      PRINT #1%, RECORD 4%, CHR$(0%);
```

The output operation has no effect on the line printer because the system software discards all NUL characters. The program maintains control of the output operation because the system does not complete the request until all previously buffered characters are printed. If an error occurs, the program can take recovery action and resume at this operation. When control passes to the next statement, the output operation is complete.

3.3.3 Clear Buffers Before Returning Control – RECORD 8%

In certain circumstances, it is advantageous for a program to terminate printing of characters already buffered for output. Because characters to be printed on a line printer reside in intermediate buffers, mere interruption of the output routine only prevents additional characters from being buffered. Normally, characters already buffered for output by the system continue printing until the buffers are clear or until an error occurs.

With the RECORD 8% option on an output request, the program tells the system to terminate the print operation and clear all pending output buffers before buffering the characters of the request. The following statement shows the procedure.

```
    10      PRINT #1%, RECORD 8%, CHR$(13%);
```

The system clears all pending output buffers and then sends the CR character to the printer. The CR character ensures that any characters in the printer hardware buffers are flushed out (by forcing them to print). After the successful completion of this statement, the printer and its buffers are clear; and the vertical and horizontal positions have been reset respectively to top of form[1] and to the left margin.

3.4 OPERATION AND ERROR HANDLING

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) or 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.

[1]The drive's internal vertical form position counter is reset to top of form. Manual paper positioning may be needed to align the form itself to its top of form position.

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

1. opens the line printer, inputs a line from the disk file, and performs output to the line printer, and
2. performs efficient error handling as described above.

```
10    !        HOUSEKEEPING
20    OPEN "DATA.DAT" FOR INPUT AS FILE 1
      /OPEN "LP0:" AS FILE 2, RECORD SIZE BUFSIZ(1%)
      /FIELD #1, BUFSIZ(1%) AS I$
      /FIELD #2, BUFSIZ(2%) AS O$
      /E% = 0%
      /ON ERROR GOTO 200
100   !        DATA MOVING LOOP
110   GET #1
      /C% = RECOUNT
      /LSET O$ = I$
140   PUT #2, COUNT C%
      /E% = 0%
      /GOTO 100
200   !        ERROR HANDLING
210        IF ERR=11 AND ERL=110          THEN RESUME 400
      ELSE IF ERR=14 AND ERL=140          THEN RESUME 300
      ELSE ON ERROR GOTO 0
300   !        PRINTER ERRORS
310   IF E% THEN 350
      !        FIRST TIME ERRORS (DATA WAS BUFFERED)
330   E% = -1%
      /GOTO 100
350   !        SUBSEQUENT ERRORS (DATA NOT BUFFERED)
360   PRINT "?LINE PRINTER ERROR?"
      /SLEEP 10%
      /GOTO 140
400   !        DONE
410   CLOSE 1, 2
32767 END
```

CHAPTER 4
TERMINALS

Several features are available to facilitate processing on keyboard devices using Record I/O statements. For hardware specific information, refer to the user's manual for the specific device on the system.

4.1 CONDITIONAL INPUT FROM A TERMINAL – RECORD 8192%

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 4.2.) 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 INPUT or INPUT LINE statements can be used in place of GET.) 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.

The RECORD 8192% option may be used with the SLEEP statement to wait for input. When a delimiter is typed at a terminal or when a receiving job has received a message, the system cancels the sleep operation. This feature can be used to determine whether the sleep operation was cancelled by terminal input or message availability. The following sample routine shows the procedure.

```
100   SLEEP 15%

110   ON ERROR GOTO 200
      \GET #1, RECORD 8192%
      \GOTO 1000
      !GOT DATA, GO PROCESS IT

200   IF ERR=13 AND ERL=110 THEN RESUME 300
       ELSE ON ERROR GOTO 0

300   !RECEIVE DATA MESSAGE IF ANY
```

If data is not available at the terminal, a message is pending. If data is available, the program can process it.

4.2 MULTIPLE TERMINAL SERVICE ON ONE I/O CHANNEL – RECORD 32767%+1%

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 job console 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 opened by 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 (or INPUT) and PUT (or PRINT and PRINT USING) statements in a special manner. The RECORD option specifies a particular action and keyboard number.

4.2.1 Multiple Terminal Service Output
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.

The RECORD option can be used with the PRINT or PRINT USING statement as well as with the PUT statement. For example, the statements

```
20   PRINT #1%, RECORD 32767%+1%+K%, Z$;
```

or

```
20   PRINT #1%, RECORD 32767%+1%+K%, USING "!!!!", Z$;
```

are valid to output the string Z$ to the unit designated by K%. With the PRINT or PRINT USING statement, the FIELD, LSET, and RSET statements associated with the PUT statement are unnecessary for moving data to an output buffer. By using PRINT or PRINT USING in place of PUT, the programmer eliminates some of the data moving code and can format the data more conveniently.

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%.

4.2.2 Multiple Terminal Service Input

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. Because this error is recoverable, the program can execute an appropriate ON ERROR GOTO routine. The system does not allow a stall on input from a specific keyboard in multiple terminal arrangements.

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 values shown in Table 4-1. If no input is pending from any terminal, the program stalls as described for the case of S%=0% in Table 4-1.

Table 4-1 Multiple Terminal RECORD Values for S%

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%=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.

4.3 TERMINAL CONTROL VIA THE MODE OPTION

Control of a terminal can be established in several ways by the MODE option in the OPEN statement. Table 4-2 summarizes the values allowed in the MODE option for a keyboard device.

Table 4-2 Summary of MODE Values for Terminals

Mode Value	Effect
1%	Binary input from a terminal
2%	Reserved for future implementation
4%	Suppress automatic carriage return/line feed at right hand margin
8%	Enable echo control for block mode simulation (turns off other modes and automatically enables MODE 4%)

The following sections describe the various MODE options.

4.3.1 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 OPEN FOR 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. The value 4096% is available in place of 1% to prevent a conflict with keyboard unit 1 in multiple terminal service output.

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.

1. The period for a WAIT statement expires (the KEYBOARD WAIT EXHAUSTED error (ERR=15) is generated)
2. Executing any input or output statement on channel zero when the user's keyboard is open for binary input
3. Executing an OPEN statement in normal mode on the device but on a different channel
4. Executing a CLOSE statement on any channel associated with a keyboard open for binary input

For condition 1, 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 2, 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. A PUT statement on channel 1, however, does not turn off binary input mode.

For condition 3, 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 4, 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.

4.3.2 Suppress Automatic CR/LF – MODE 4%

RSTS/E normally performs a carriage return/line feed operation when the right hand margin of a terminal is to be exceeded. (TTYSET sets the right hand margin via the width characteristic.) This automatic operation is suppressed by opening the terminal with the MODE 4% option as follows.

```
OPEN 'KB13:' AS FILE 1%, MODE 4%
```

The terminal on keyboard line 13 is opened on channel 1 with suppress CR/LF mode. All terminals assigned by the job but not opened are placed in the same mode. (This action follows the multiple terminal service rules.) All slave terminals thereby have the same control characteristics as the master terminal.

The suppression remains in effect until the terminal is either closed or opened again without MODE 4%. All slave terminals maintain the suppression mode until the master terminal is either closed or opened again without MODE 4%.

MODE 4% is normally used for echo control and is automatically enabled with the MODE 8% option described in Section 4.3.3.

4.3.3 Echo Control and Block Mode Simulation – MODE 8%[1]

RSTS/E allows a full duplex terminal to simulate block mode operation by special echo control mode. In block mode operation, all data entered on a screen is sent to a computer in one operation. Before the data is sent, it can be edited locally and its latest state be visible on a screen. In block mode simulation, echo control is provided for a discrete field on a screen defined by the user program. The echo control maintains the integrity of data outside of the declared field. Although each field is sent to the computer separately, the program can maintain the appearance that the entire screen is being sent as a block to the computer. Because each field is processed by the computer, more sophisticated error checking of input data can be performed than is possible in a true block mode device.

With echo control, the program can declare a field on a video terminal and define a special character for character deletion sequence within the field. The special character, called the paint character, is refreshed on the screen if characters typed in the field are deleted before input to the program. The paint character, therefore, maintains the visible extent of a field. The system automatically maintains any declared paint character. The program can output

[1]Echo control is an optional feature of the RSTS/E monitor and may not be available on all systems.

prompting messages on the screen, accept input from one field at a time, and format the data for processing. This feature can be used on hard copy terminals although it was designed primarily for scope terminals.

Echo control is enabled on a terminal by the MODE 8% option in the OPEN statement as follows.

```
    OPEN 'KBn:' AS FILE 1%, MODE 8%
```

The keyboard designated by n is opened on channel 1 in echo control mode. A nonzero channel is required. All terminals assigned by the job but not opened are also placed in echo control mode. (This action follows the multiple terminal service rules described in Section 4.2. Thus, all slave terminals are in the same mode as the master terminal.)

MODE 8% also turns off all other MODE options in effect and forces on MODE 4%.

Echo control remains in effect until one of the following conditions is met:

1. a CLOSE is performed on the channel,
2. the terminal is opened again without MODE 8%, or
3. any input or output is performed on channel 0 (the job's console terminal).

All slave terminals maintain echo control until one of the above listed conditions is met.

In echo control mode, the parity bit is stripped from all characters. All characters returned to the user have ASCII values in the range 1 to 127. Table 4-3 summarizes how these characters are treated in echo control mode. Synchronization and editing characters are not passed to the program. Delimiters are passed to the program but are never echoed.

Table 4-3 Echo Control Mode Character Set

Class	ASCII Code	Code Returned to User	Comments
Ignored	0	–	Used as filler for timing
Delimiters	private	?	Private delimiter
	3	3	↑C (CTRL/C combination)
	4	4	↑D (CTRL/D combination)
	10	10	Line feed
	12	12	Form feed
	13	13,10	Carriage return (Line feed is appended)
	26	26	↑Z (CTRL/Z combination); generates ERR = 11
	27	27	If 'NO ESC SEQ' is in effect and escape is received, 27 is returned to user and is treated as a delimiter. If 'ESC SEQ' is in effect, escape triggers an escape sequence. The escape sequence is returned to user (see Section 4.4.2) and the whole sequence is considered the delimiter.
	125	27 or 125	If 'NO ESC' is in effect, 125 is translated to escape (27).

(continued on next page)

Table 4-3 (Cont.) Echo Control Mode Character Set

Class	ASCII Code	Code Returned to User	Comments
Delimiters (cont.)			If 'ESC' is in effect, 125 is data.
	126	27 or 126	If 'NO ESC' is in effect, 126 is translated to escape (27). If 'ESC' is in effect, 126 is data.
Editing	127	–	Rubout (DEL character); on scope terminals, generates a backspace followed by the paint character and another backspace; on hard copy terminals, echoes deleted characters between backslashes.
	21	–	↑U (CTRL/U combination); repeatedly simulates Rubout until no characters remain in field.
Data	32-95	32-95	Normal 64-character graphic set
	96-126	64-94	If 'NO LC INPUT', lower case are translated to upper case.
	96-126	96-126	If 'LC INPUT', lower case are returned to user.
Synchro-nization	17	–	XON (CTRL/Q combination). Resume suspended output (if 'STALL').
	19	–	XOFF (CTRL/S combination). Suspend output (if 'STALL')
Other	1,2,5,6, 7,8,9,11, 14-16,18, 20,22-25, 28-31	–	Echoed as BEL (code 7) but otherwise ignored.
	17,19	–	If 'NO STALL', synchronization characters are treated as other.

When the terminal is opened in echo control mode, no echoing is done until a field is declared on the channel. Declaring a field has the following effects.

1. establishes field size which is the maximum number of characters the field can hold.
2. specifies how overflow characters are handled. Two methods are available.
   a. Normal. A field is terminated by receiving a delimiter. Any characters received in excess of the field size are treated as other (see Table 4-3) and echoed as BEL characters.
   b. Keypunch. A field is terminated by either receiving a delimiter or by entering the nth character in an n-character field. If the field is terminated by size (receiving the maximum number of characters allowed) rather than by receiving a delimiter, a form feed (code 12) is appended to the field. Any excess characters entered are not echoed but are retained as input for the next field.

3. defines a special character to be echoed for character deletion sequences. The default is the space character which actually erases a visible character on a video screen. A character like underscore, however, can be used to indicate, or paint, the field. Accordingly, an editing character (↑U or DEL) causes the defined paint character to be echoed in place of the default space character. This action maintains the visual indicator of the field during any character deletion sequence.

To declare a field, a special form of the PUT or PRINT statement is executed on the channel on which the terminal is open with MODE 8%. The RECORD 256% and COUNT N% options are used with the PUT statement to declare the field as follows.

```
    PUT #C%, RECORD 256%, COUNT N%
```

where N% indicates how many bytes in the buffer of channel C% are to be considered for the field declaration. Only the first two bytes in the buffer are significant.

byte 1 (required)	contains the field size and overflow handling information. The size must be between 1 and 127. Attempting to declare a size of 0 generates the ILLEGAL BYTE COUNT FOR I/O error (ERR = 31). 128 added to the field size indicates keypunch overflow handling is to be used instead of normal overflow handling.
byte 2 (optional)	indicates the ASCII value of the paint character. If this byte is 0 or is not used, space is the default paint character.
byte 3 and following	ignored

COUNT 1% in the PUT statement therefore specifies that only the first byte in the buffer is to be used to declare the field. Space becomes the paint character by default. COUNT 2% indicates that the first two bytes in the buffer are to be used. Omitting the COUNT option from the PUT statement, using COUNT 0% and using a COUNT value greater than 2 are equivalent to specifying COUNT 2%. In all cases, characters in the buffer beyond the second byte are ignored.

The PRINT statement can also declare a field by a method similar to that of the PUT statement. The RECORD 256% modifier in the PRINT statement indicates the field declaration. A one- or two-byte string replaces the COUNT option to indicate field parameters. The following forms are valid.

```
    10 PRINT #C%, RECORD 256%, CHR$(M%+S%);
```

or

```
    10 PRINT #C%, RECORD 256%, CHR$(M%+S%)+"P";
```

or

```
    10 PRINT #C%, RECORD 256%, CHR$(M%+S%)+CHR$(P%);
```

where:

C%	is the nonzero channel open with MODE 8%
M%	is the overflow handling mode M%=128% for keypunch M%=0% for normal

S%	is the field size between 1 and 127
+	concatenates the second (optional) byte
P	is the ASCII paint character
P%	decimal code for paint character (for example, underline is CHR$(95%))
;	terminates the string

Usage of the PRINT statement to declare a field saves user space because the statements to define and load a buffer are eliminated.

After the field is declared, the system enables echoing. The declaration makes the field active. Characters are then echoed until the field is filled. Subsequent characters are handled according to the overflow mode in effect for the field. When a delimiter is received (or the nth character for an n-character keypunch field is received), the field is deactivated. Echoing is again disabled. Characters typed after the field is deactivated are retained until the next field is declared.

Attempting to declare a field when one is currently active generates the ACCOUNT OR DEVICE IN USE error (ERR = 3) unless nothing has been entered in the active field. The SYS call to cancel type ahead deactivates an active field.

The 256% value can be used with other values in the RECORD option of the PUT or the PRINT statement for multiple terminal service operations. The ability to combine RECORD values enables the program to declare a field for either the master or a slave terminal. Fields need not be declared on all terminals, only on those terminals from which input is solicited. If the program attempts to input data without declaring a field on any terminal, the job will be locked in an input wait state. The WAIT statement may be used to recover and continue execution.

The following operations are a recommended sequence to execute when interacting with a scope terminal in echo control mode.

1. Open any terminal on a nonzero channel with MODE 8%.
2. Execute SYS call 11 to cancel type ahead. This action discards any unsolicited input which would be echoed automatically after a field is declared.
3. Position the cursor to top of screen and clear the screen.
4. Print any prompting text and display paint characters in all fields. (The program must initially display the paint characters which will be maintained by the terminal service during any deletion sequences.)
5. Position the cursor to the beginning of the first field (by direct cursor addressing).
6. Declare the field with the desired size and designate a paint character equivalent to the one displayed.
7. Execute the GET statement to retrieve input. The INPUT, INPUT LINE and MAT INPUT statements recognize only the standard BASIC-PLUS delimiters (carriage return, line feed and form feed) and should not be used for input operations. Additionally, the GET statement allows recognition of the private delimiter.
8. Extract data from the buffer and store it for processing.
9. Continue positioning the cursor, declaring fields, retrieving input, and extracting data as required.

The sequence is slightly different when working with a hard copy terminal.

1. Open the terminal on a nonzero channel with MODE 8%.
2. Execute SYS call 11 to cancel type ahead.
3. Position the paper at top of form. (If the terminal has hardware top of form, print a form feed. Otherwise, print several line feeds.)
4. Print any prompting text for the first field.
5. If the terminal can backspace, and has the underline character, paint the field with underlines and print the appropriate number of backspaces to fix the printing position at the start of the field.
6. Declare the field with the desired size and overflow handling mode. Do not declare a paint character because it has no effect on a hard copy terminal.

7. Execute the GET statement to retrieve input. The INPUT, INPUT LINE and MAT INPUT statements should not be used.
8. Extract data from the buffer and store it for processing.
9. Position the paper and printing mechanism for the next field by printing carriage return, line feeds, and spaces as required. Use only one field per line because characters removed during a deletion sequence are echoed which can cause the next intended field to be used.
10. Repeat the sequence from step 4 until all fields are satisfied.

4.4 ESCAPE SEQUENCES

4.4.1 Output Escape Sequences

When sending an escape sequence to a terminal, use the value CHR$(155) for the escape character rather than CHR$(27). The system translates CHR$(27) to CHR$(36) 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$(27) – acts as a delimiter. The system echoes a CHR$(36), which is the dollar sign ($) character.

In ESC SEQ mode, however, an incoming escape character CHR$(27) merely sets a flag indicating that an escape sequence follows. No character is echoed for CHR$(27) nor are characters in the sequence other than control characters echoed. The characters within the escape sequence are handled as follows.

1. ASCII control characters (CHR$(0) through CHR$(31) and CHR$(127)) are processed first. Although embedded in an escape sequence, their functions remain unchanged. (For this reason, control characters are typically not used in escape sequences.) The control character CHR$(27) itself triggers the start of a new escape sequence.

NOTE

The function of control characters DEL CHR$(127) and CTRL/U CHR$(21) do change. They are discarded and not passed to the user.

2. Any number of filler characters (CHR$(32) through CHR$(47)) can follow the triggering ESC character.
3. Within an escape sequence, any normal data conversion, such as translating lower case to upper case, is not done.
4. If an escape sequence has been triggered but not terminated, the system continues to process it.
5. After the escape sequence is terminated, normal data conversions are done.

Escape sequences are terminated by one of the following forms:

1. Y (CHR$(89)) followed by two trailing characters

   <ESC><n fillers><Y><y-addr><x-addr>

   (This form is used by some terminals for direct cursor addressing.)
2. 0 (CHR$(79)), P (CHR$(80)), or ? (CHR$(63)) followed by one trailing character

   <ESC><n fillers><0><modifier>
   <ESC><n fillers><P><modifier>
   <ESC><n fillers><?><modifier>

(The modifier is any character other than a control character.)

3. No trailing characters

   < ESC >< n fillers >< other character >

   (Other character is any character other than a control or filler character.)

Only the above listed forms or another ESC character terminates the escape sequence.

The escape sequence is passed to the program as follows:

1. The triggering ESC character is replaced by CHR$ (128)
2. The escape sequence characters with no normal data conversion.
3. CHR$ (155) indicating the termination of the escape sequence.

For example, if the escape sequence:

< ESC >< / >< 0 >< A>

is received, the system passes to the program the following data:

< CHR$ (128) >< / >< 0 >< A >< CHR$ (155) >

The ESC character is translated to CHR$ (128). The slant character is a filler and is passed to the program. The 0 character signals that one trailing character follows. Receiving the A character, which is not a control or a filler character, terminates the escape sequence. The system passes the CHR$(155) character to the program to signal the termination of the escape sequence.

4.5 PSEUDO KEYBOARD OPERATIONS

A pseudo keyboard is a non-physical device that has the characteristics of a terminal but has no physical device associated with it. Like a terminal, a pseudo keyboard has both input and output buffers to which user programs send input and from which they extract output. By using a pseudo keyboard, one job can control other jobs on the system. Pseudo keyboards are particularly useful for BATCH operations, so that a terminal need not be tied up unnecessarily.

At system generation time, the system manager sets the number of pseudo keyboards on the system. The system denotes a pseudo keyboard by a device designator of PK and associates each pseudo keyboard with a keyboard unit number (but not with a physical terminal device.) For example, the system may associate PK5: with KB8:, although no physical keyboard number 8 exists. The output buffer of the pseudo keyboard is the input buffer for the associated keyboard unit and vice versa.

Use of a pseudo keyboard involves a controlling job and a controlled job. The controlling job initiates operations by performing output to the pseudo keyboard unit. It may run LOGIN and use both RSTS/E and program commands to control the job. The controlling job uses the pseudo keyboard to perform input to and extract output from the controlled job (which runs on KBn: associated with PKm:). The controlled job does input and output operations on its own keyboard, KB:. In effect, the controlled job does not know it is using a pseudo keyboard; only the controlling job specifically uses a pseudo keyboard. Figure 4-1 shows the interaction between the controlled and controlling jobs.

The system transfers data to the pseudo keyboard in full duplex mode. This mode means that strings transmitted by PUT statements are echoed in the output buffer of the associated keyboard unit. The echo is then available to the controlling job by GET, INPUT, or INPUT LINE statements. In addition, a CR character (CHR$(13)) sent to the PK input buffer returns from the KB buffer as a CR and LF sequence.

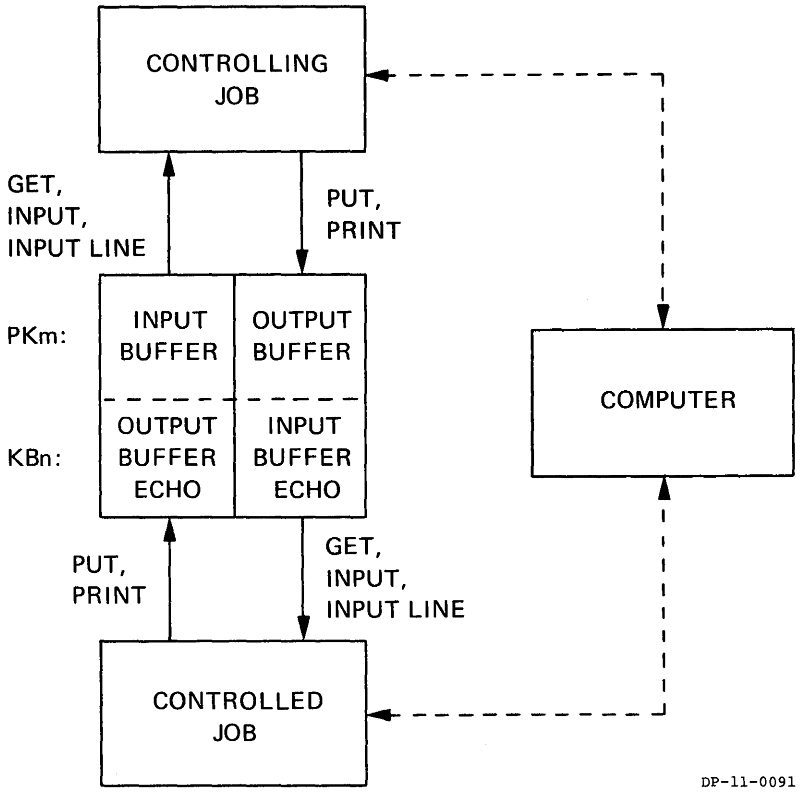


Figure 4-1 Pseudo Keyboard Operations

4.5.1 Pseudo Keyboard I/O

A controlling job first accesses pseudo keyboard n by the OPEN statement with the device designator PKn: as follows:

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

The OPEN statement associates pseudo keyboard unit 0 with internal channel 1. The system ignores specifications of FOR INPUT and FOR OUTPUT for pseudo keyboards. If the device type or the unit number specified was not configured on the system, the NOT A VALID DEVICE error (ERR=6) is generated. If the device is assigned to or opened by another job, the DEVICE NOT AVAILABLE error (ERR = 8) occurs. Otherwise, the program can perform GET and PUT operations on the pseudo keyboard through the I/O buffer for channel 1.

To obtain data output from the controlled job, the controlling program executes a Record I/O GET statement on the proper internal channel. For example, the following makes data from the pseudo keyboard available in channel 1 buffer for the controlling job.

```
100 GET #1%
```

The response to a GET statement is immediate. The system never stalls the controlling program pending data availability. This means that when the system executes a GET statement, it returns the contents of the buffer to the controlling job. The contents of that buffer could be a single message, multiple messages, or a message fragment. If no input is available, an END OF FILE ON DEVICE (ERR = 11) error occurs.

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 number of the internal channel on which the PK device is open. The value of C% is the number of bytes the program sends from the buffer to the pseudo keyboard. Without the COUNT option, the PUT statement transmits either the number of bytes specified in the RECORDSIZE option of the OPEN statement or the default of 128 bytes (if no RECORDSIZE was specified).

The program must send data from the buffer in the same format that the user 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%)) as a terminator and the value C% must reflect the CR character in the total number of bytes sent. The system automatically 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 a specific PUT statement. R% is an 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 integer R%. The system executes the PUT statement depending upon whether certain bits in the value are on or off.

Figure 4-2 shows the flow of actions the system performs by testing each bit in the RECORD R% option. Bit 0 (value = 1) establishes whether the system attempts to send data or performs tests before it sends data.

Bit 1 (value = 2) tests 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 on) or simply returns control to the controlling program. Finally, 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 job on the pseudo keyboard device. RECORD 1% sends data to the keyboard and thus can send LOGIN program commands. A GET statement retrieves 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 system program. A subsequent GET statement retrieves the echo and the PASSWORD: message printed by LOGIN. The controlling program then can send the proper password string and ensure that LOGIN accepts it.

Once the controlled job is running, the controlling program sends system commands, program commands, and program responses to the PK device by using various RECORD options in PUT statements. The program should send only one line at a time and retrieve each program or system response separately. The program should not transmit multiple lines because this fills up small buffers. For the same reason, the user should not type ahead. In addition, 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 2 and 4) 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 controlling program must then force a CTRL/C combination to the controlled job. Otherwise, control returns to the controlling job, which can then transmit a system command.

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 returns to the controlling job. If not, the system generates error number 3. If the controlling job logs out or closes the pseudo keyboard before running LOGOUT for the controlled job, the controlled job detaches from the pseudo keyboard.

A program can force data to the keyboard side of a pseudo keyboard, if the pseudo keyboard side of the associated keyboard unit is currently open; data forced to the keyboard unit side is placed in the input buffers of that unit. If the pseudo keyboard side is not open, data forced to the keyboard unit side is discarded by the system.

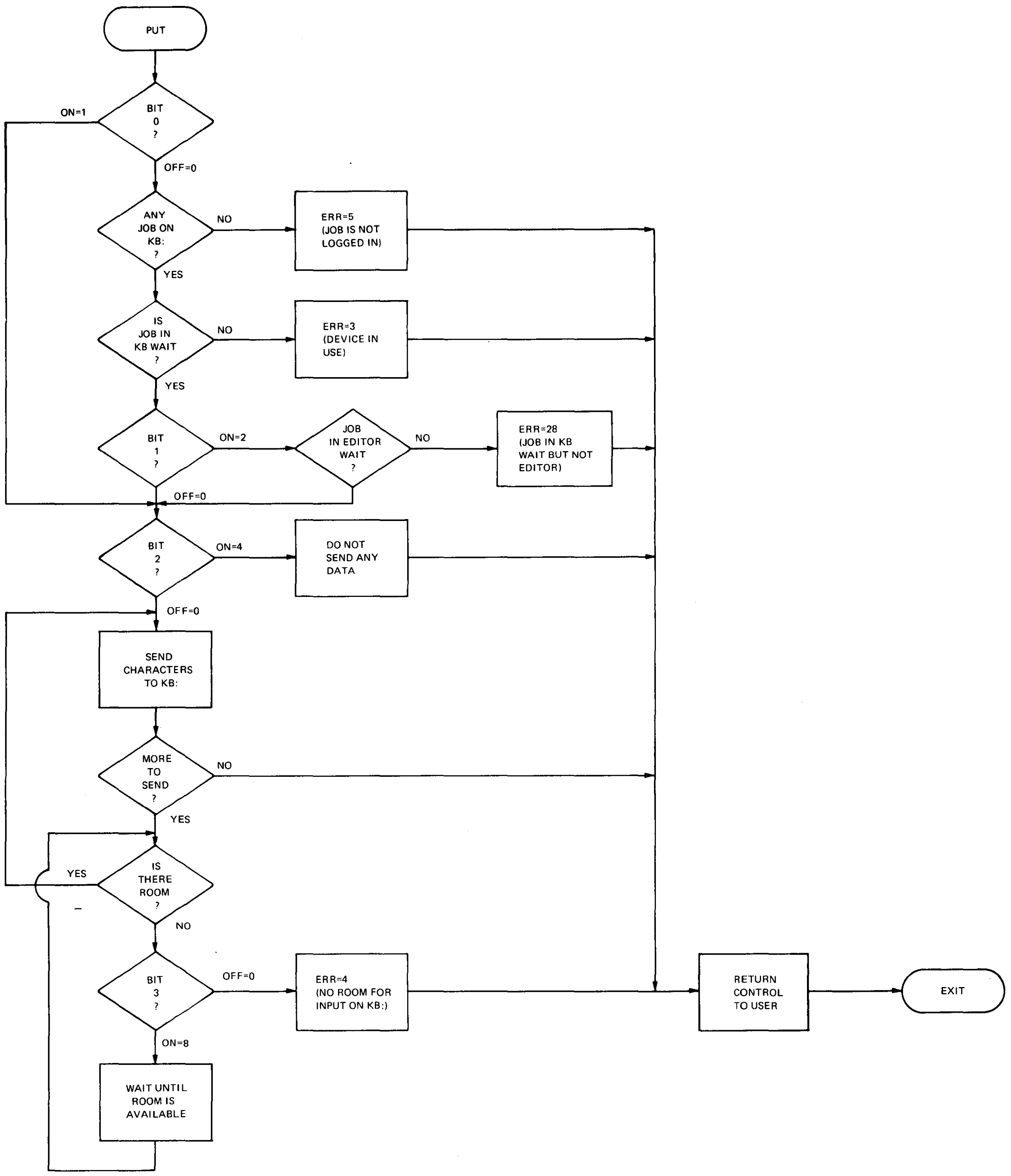


Figure 4-2 PUT Statement Actions for Pseudo Keyboard Output

CHAPTER 5
DECTAPE

5.1 FILE STRUCTURED DECTAPE

The user program indicates file structured processing on DECtape by including a file name with the device specification in the OPEN statement. Up to 12 files can be open for read access simultaneously on a DECtape drive. Only one file, however, can be open for write access on a DECtape drive. An attempt to open a second file for write access while one is currently open for write access generates the TOO MANY FILES OPEN ON UNIT error (ERR=17).

When a file is opened on DECtape, RSTS/E implicitly assigns the unit to the job performing the OPEN operation. Another job attempting to access the DECtape receives the DEVICE NOT AVAILABLE error (ERR=8). For the job performing the OPEN operation, BASIC-PLUS creates a 510-byte buffer. Only 510 bytes are usable in a file structured DECtape block because the system treats the remaining 2 bytes as a pointer to the next block in the file. (Refer to the figure in Section 5.2.) GET and PUT statements read and write successive blocks on the tape. The RECORD option cannot be used to access blocks in the file in a random manner.

If the RECORDSIZE option is specified in the OPEN statement, a buffer of the value given in the option is created. For a buffer size less than 510 bytes, a GET statement returns that many bytes from the first part of 510-byte block. A PUT statement writes one block and fills the remainder of the 510 bytes with NUL characters. For a buffer size greater than 510 bytes, a GET statement reads one block of 510 bytes and a PUT statement generates the ILLEGAL BYTE COUNT FOR I/O error (ERR=31).

5.2 NON-FILE STRUCTURED PROCESSING

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. BASIC-PLUS creates a 512-byte buffer. 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:

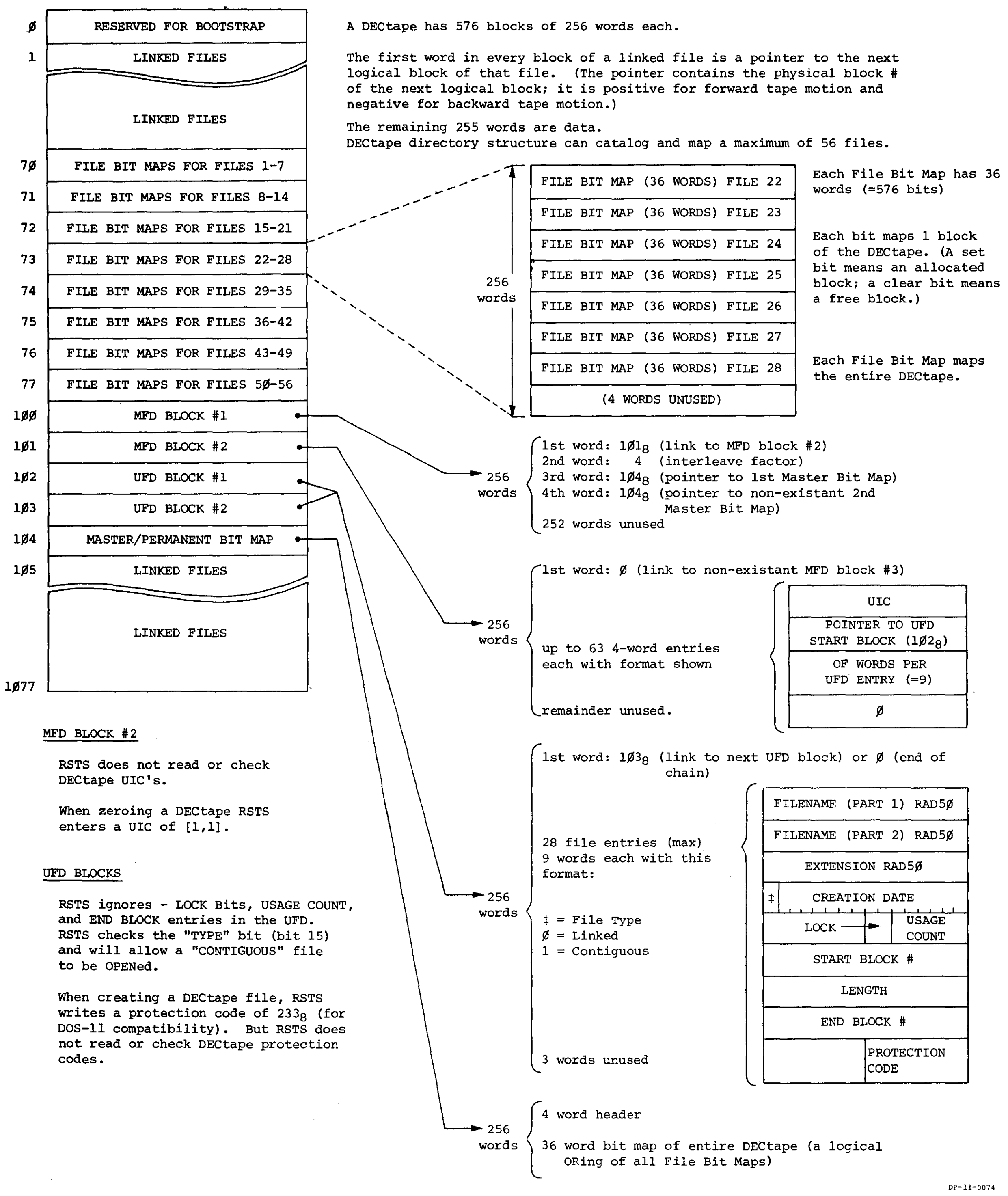
```
200 GET #1%, RECORD -4%
```

reads block 4 of the file opened on channel 1% backwards. The maximum block number is 578.

In writing non-file structured DECtape files, the user can specify how blocks should be accessed. Whenever possible, 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.

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.

The link pointer is either positive or negative. Negative indicates the block was written in the reverse direction. Positive indicates the block was written in the forward direction. Use the negative value in RECORD option to access a block written backwards.



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
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)
256 words
256 words
256 words
256 words
256 words
UIC
POINTER TO UFD START BLOCK (1Ø2₈)
OF WORDS PER UFD ENTRY (=9)
Ø
FILENAME (PART 1) RAD5Ø
FILENAME (PART 2) RAD5Ø
EXTENSION RAD5Ø
‡ CREATION DATE
LOCK
USAGE COUNT
START BLOCK #
LENGTH
END BLOCK #
PROTECTION CODE
DP-11-0074

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.

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.

1st word: 1Ø1$_8$ (link to MFD block #2)
2nd word: 4 (interleave factor)
3rd word: 1Ø4$_8$ (pointer to 1st Master Bit Map)
4th word: 1Ø4$_8$ (pointer to non-existant 2nd Master Bit Map)
252 words unused

1st word: Ø (link to non-existant MFD block #3)
up to 63 4-word entries each with format shown
remainder unused.

1st word: 1Ø3$_8$ (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

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:

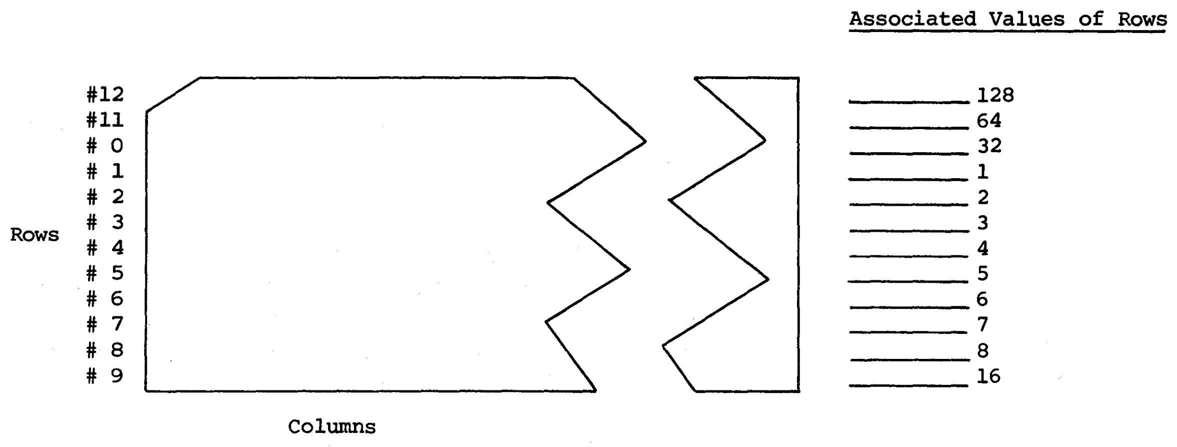
```
100   OPEN "CR:" AS FILE 1%
      \INPUT LINE #1%, A$
      \PRINT LEN(A$)
      \PRINT ">" A$ "<"
32767   END

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. With INPUT or INPUT LINE statements, any columns containing illegal punches are stored as RUBOUT (ASCII 127) codes. 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 255) codes. By checking the characters for RUBOUT, 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	Ø	9	8	1-7		
VALUE	128	64	32	16	8	4	2	1

DP-11-0075

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 values of the punched row positions, as shown in Figure 6-2.

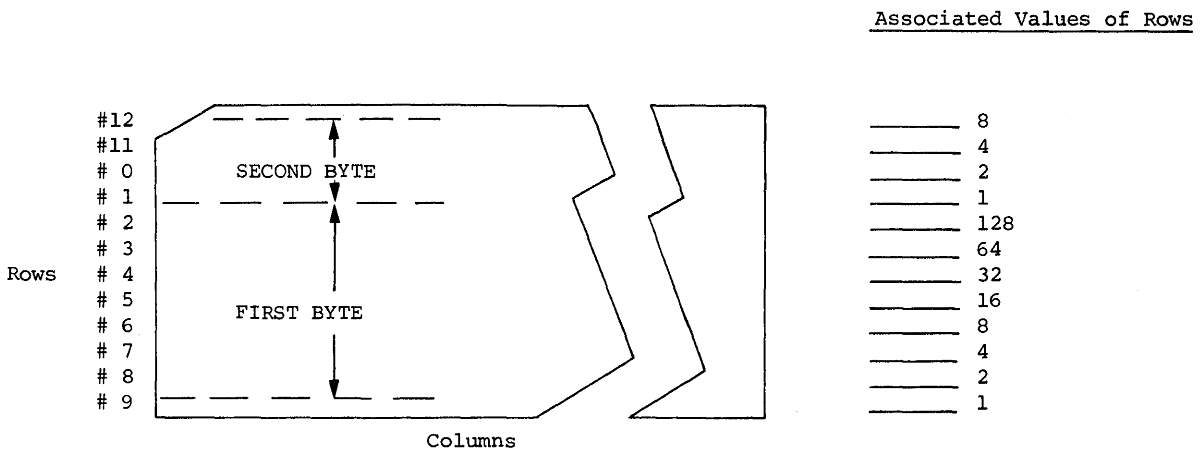
6.4 SETTING READ MODES

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 6-1. The default mode is 0 (ASCII). When a MODE or RECORD option is used, an explicit value as shown in Table 6-1 must be specified; failure to do so results in an error message.

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.



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

DP-11-0076

Figure 6-2 Binary Read Mode

Table 6-1 Specifying Read Modes on Card Reader

	MODE or RECORD Option	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

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

      \GET #1%, RECORD 256%                             ASCII

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

```
                                                  Specified Read Mode
                                                     at This Point

        \GET #6%                                     ASCII

        \GET #6%, RECORD 258%                        Binary

        \CLOSE 6%

    32767 END
```

After line 100, above, sets the read mode to Hollerith, the next line overrides it, setting the read mode to ASCII temporarily. When the following line 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. The next line is executed without a RECORD option, so the next card is read also in ASCII read mode. Closing a CR file, 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 RSTS/E implementation of the BASIC language. As such, the calls are system dependent and have calling formats 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 nonprivileged. The privileged calls can be used only by a privileged user or by a privileged program. The nonprivileged calls can be used by anyone and are completely safe in the sense that their misuse can do no damage to other programs or to the system.

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 eleven, 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). Table 7-1 summarizes the codes and their usages. The subsequent pages describe the usage, calling format, and purpose of the calls.

Table 7-1 SYS System Function Codes

Function Code (F)	Usage
0	Cancel ↑O effect on terminal
1	Enter tape mode on terminal
2	Enable echoing on terminal
3	Disable echoing on terminal
4	Enable delimiterless character input mode (ODT submode) on terminal
5	Exit with no prompt message
6	SYS call to the file processor
7	Get core common string
8	Put core common string
9	Exit and clear program
10	Reserved for special implementations
11	Cancel all type ahead

7.1.2 General SYS System Function Calls

7.1.2.1 Cancel ↑O Effect on Terminal – Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(0%), the cancel ↑O effect code
2	CHR$(N%) where N% is the number (between 0 and 12) of the channel on which the system executes the call. If this byte is not specified, channel 0 is used.
3	CHR$(K%) where K% is the number (between 0 and 127) of the keyboard assigned but not open by the job. This follows the multiple terminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel specified in byte 2. If this byte is not specified, the keyboard effected is the one open on the channel specified in byte 2.

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 terminal specified. The terminal is selected by channel number given in byte 2. (The terminal must be open on that channel.) If a slave terminal is to be used, byte 2 must be a nonzero channel number (on which the master terminal is open) and byte 3 denotes the keyboard number of the slave terminal. See the *RSTS/E System User's Guide* for a description of the CTRL/O combination.

7.1.2.2 Enter Tape Mode on Terminal – Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(1%), the enter tape mode code
2	CHR$(N%) where N% is the number (between 0 and 12) of the channel on which the system executes the call. If this byte is not specified, channel 0 is used.
3	CHR$(K%) where K% is the number (between 0 and 127) of the keyboard assigned but not open by the job. This follows the multiple terminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel specified in byte 2. If this byte is not specified, the keyboard effected is the one open on the channel specified in byte 2.

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/E System User's Guide*. The terminal is selected by channel number given in byte 2. (The terminal must be open on that channel.) If a slave terminal is to be used, byte 2 must be a nonzero channel number (on which the master terminal is open) and byte 3 denotes the keyboard number of the slave terminal.

7.1.2.3 Enable Echoing on Terminal – Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(2%), the enable echoing code
2	CHR$(N%) where N% is the number (between 0 and 12) of the channel on which the system executes the call. If this byte is not specified, channel 0 is used.
3	CHR$(K%) where K% is the number (between 0 and 127) of the keyboard assigned but not open by the job. This follows the multiple terminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel specified in byte 2. If this byte is not specified, the keyboard effected is the one open on the channel specified in byte 2.

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

Discussion:

This code cancels the effect of SYS calls with Codes 1 and 3. The terminal is selected by channel number given in byte 2. (The terminal must be open on that channel.) If a slave terminal is to be used, byte 2 must be a nonzero channel number (on which the master terminal is open) and byte 3 denotes the keyboard number of the slave terminal.

7.1.2.4 Disable Echoing on Terminal – Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(3%), the disable echoing code
2	CHR$(N%) where N% is the number (between 0 and 12) of the channel on which the system executes the call. If this byte is not specified, channel 0 is used.
3	CHR$(K%) where K% is the number (between 0 and 127) of the keyboard assigned but not open by the job. This follows the multiple terminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel specified in byte 2. If this byte is not specified, the keyboard effected is the one open on the channel specified in byte 2.

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

Discussion:

This call prevents the system from echoing information typed at the terminal. As a result, information such as a password is kept secret but accepted as valid input by the system. The terminal is selected by channel number given in byte 2. (The terminal must be open on that channel.) If a slave terminal is to be used, byte 2 must be a nonzero channel number (on which the master terminal is open) and byte 3 denotes the keyboard number of the slave terminal.

7.1.2.5 Enable Delimiterless Character Input Mode (ODT Submode) on Terminal – Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(4%), the enable single character input mode code
2	CHR$(N%) where N% is the number (between 0 and 12) of the channel on which the system executes the call. If this byte is not specified, channel 0 is used.
3	CHR$(K%) where K% is the number (between 0 and 127) of the keyboard assigned but not open by the job. This follows the multiple terminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel specified in byte 2. If this byte is not specified, the keyboard effected is the one open on the channel specified in byte 2.

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

Discussion:

This call allows less than a full line 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 or CTRL/D combination has been typed before accepting input. In delimiterless character mode, one or more characters typed at the terminal are passed immediately to the program by the next keyboard input request statement without waiting for a delimiting character.

This function must be enabled prior to every input request statement that immediately passes characters 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". The terminal is selected by channel number given in byte 2. (The terminal must be open on that channel.) If a slave terminal is to be used, byte 2 must be a nonzero channel number (on which the master terminal is open) and byte 3 denotes the keyboard number of the slave terminal.

7.1.2.6 Exit with No Prompt Message – Not Privileged

Data Passed:

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

Data Returned: No data is returned.

Discussion:

This type of exit does not clear the program from memory and thus allows the user to continue running the program. The following are the specific effects.

1. No files are closed
2. The current program state is saved (to allow a continue)
3. No prompting message is generated
4. The BASIC-PLUS editor waits for a command

7.1.2.7 FIP Function Call – Both Privileged and Not Privileged

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

7.1.2.8 Get Core Common String – Not Privileged

Data Passed:

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

Data Returned: The target string is the contents of the job 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. This call does not alter the contents of the core common area. Refer to SYS function code 8.

7.1.2.9 Put Core Common String – Not Privileged

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 and Clear Program – Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(9%), the exit and set up NONAME code
2-3	First 3 characters of the name of the run time system, in Radix-50 format, to which control is passed.
4-5	Last 3 characters of the name of the run time system, in Radix-50 format, to which control is passed.

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

Discussion:

This call clears the current program from memory and returns control to the user's private run time system. It also closes all channels without cleaning up partial buffers. (That is, any I/O in progress is not completed). This is the proper way of stopping a program that is not to be rerun. Such programs are those that terminate on an error and are privileged. The same action is performed by the command NEW NONAME.

If bytes 2 through 5 specify a run time system, the call transfers control to that run time system and establishes it as the job's private default run time system. If bytes 2 through 5 are not specified, control is transferred to the job's private default run time system.

The run time system to which control is returned prints its prompting message. For the BASIC-PLUS run-time system, two prompts are possible. If the job is logged into the system, BASIC-PLUS prints carriage return, line feed, and Ready followed by one carriage return and two line feeds. If the job is not logged in, BASIC-PLUS prints carriage return, line feed and BYE followed by one carriage return and two line feeds.

7.1.2.11 Cancel All Type Ahead – Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(11%), the cancel type ahead code
2	CHR$(N%) where N% is the number (between 0 and 12) of the channel on which the system executes the call. If this byte is not specified, channel 0 is used
3	CHR$(K%) where K% is the number (between 0 and 127) of the keyboard assigned but not open by the job. This follows the multiple terminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel specified in byte 2. If this byte is not specified, the keyboard effected is the one open on the channel specified in byte 2.

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

Discussion:

This call clears all unread, pending input from a terminal's buffers. This cancels any input typed in advance of programmed solicitation of input. The application of this call is mainly for echo control operations where echoing of unsolicited input ruins the visual indication in painted fields. Refer to Section 4.3.3 for the discussion of controlling echo and declaring a field on a screen to have a special paint character.

The terminal is selected by channel number given in byte 2. (The terminal must be open on that channel.) If a slave terminal is to be used, byte 2 must be a nonzero channel number (on which the master terminal is open) and byte 3 denotes the keyboard number of the slave terminal.

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 (normally meaningless to general users)
3-30	Data returned

NOTE

Except for the message send/receive calls, 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 more convenient 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)
```

NOTE

If any part of a parameter string is documented as Not used, it is suggested that those bytes be filled with NUL characters. The STRING$(n,0%) function (where n is the number of NUL characters needed) can be used to generate a string of proper length. If future versions of RSTS/E use these currently unused bytes, the value 0% will not force current coding to change to maintain old functionality.

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 (a 30-element 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%)
        \J% = 4%
        \A%(I%) = 0% FOR I% = 0% TO 30%
        \A%(0%) = 30%
        \A%(1%) = 6%
        \A%(2%) = 8%
        \A%(3%) = J%
        \A%(27%) = 0%
        \A%(28%) = 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 more convenient 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 shown in Figure 7-1.

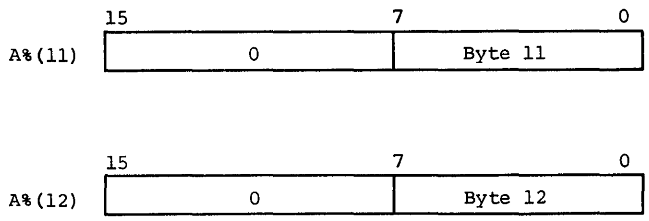


Figure 7-1 Integer Representation of CHANGEd Characters

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 shown in Figure 7-2.

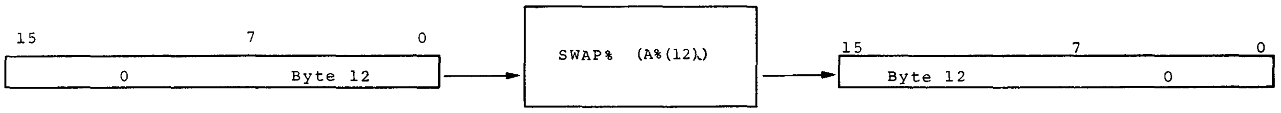


Figure 7-2 Reversal of Bytes by SWAP%( ) Function

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. Refer to Section 9.5 for a more detailed description of the SWAP% function and its use with the CVT functions.

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

7.2.3.1 Project-Programmer Number – 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 and (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

7.2.3.2 Integer (2-byte) Numbers – 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 and (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. Refer to Section 9.5 for a discussion of the SWAP% function with the CVT functions. 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 puts 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%))
```

7.2.3.3 Unsigned Integer (2-byte) Numbers – 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. Because 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.* in the statement with SWAP%( ) causes the expression to be first evaluated as a normal integer expression, and then changed to a floating point value. This operation is not desirable because 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       Q = Q - 32768.
           \ Q% = Q% EQV 32767%
           \ A%(5%) = Q% AND 255%
           \ A%(6%) = SWAP% (Q%) AND 255%
```

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

7.2.3.4 File Name String Scan Format – 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%)
             \A$="DK0:INVENT.DAT[10,20]"
             \CHANGE SYS(CHR$(6%)+CHR$(-10%)+A$) TO A%
             \A%(0%)=30%
             \A%(1%)=6%
             \A%(2%)=17%
             \A%(3%),A%(4%)=0%
             \CHANGE A% TO A$
             \CHANGE SYS(A$) TO A%
    32767    END
```

Many calls require a filename, password, pack identification label or other 6-character string to be passed as 2 words in Radix-50 format. The filename string scan call is the only means provided to convert the string to the proper format. Section 7.2.4.1 describes how this conversion is done.

NOTE

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 7-2 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 page shown beside the name in the table.

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 nonprivileged users.

Table 7-2 FIP SYS Calls (by Sub-function Code)

Function Code (F0)	Privileged[1] Status	Function Name	Page
-25	No	Read or Write Attributes	7-110
-24	Yes	Add/delete CCL command	7-53
-23	No	Terminating file name string scan	7-19
-22	Yes	Set special run priority	7-57
-21	Yes	Drop/Regain (temporary) privileges	7-59
-20	Yes	Lock/unlock job in core	7-58
-19	Yes	Set number of logins	7-84
-18	Yes	Add run-time system	7-104
	Yes	Remove run-time system	7-106
	Yes	Load run-time system	7-107
	Yes	Unload run-time system	7-109
-17	No	Name run-time system	7-103
-16	Yes	System shutdown	7-38
-15	Yes	Accounting dump	7-88
-14	Yes	Change system date/time	7-39
-13	Yes	Change priority/run burst/job size	7-55
-12	No	Get monitor tables – Part II	7-100
-11	Yes	Change file backup statistics	7-68
-10	No	Filename string scan	7-19
-9	Yes	Hangup a dataset	7-40
-8	No	FCB/DDB Information	7-101
-7	No	CTRL/C Trap enable	7-36
-6	Yes[2]	Poke memory	7-83
-5	Yes	Broadcast to terminal	7-41
-4	Yes	Force input to terminal	7-42
-3	No	Get monitor tables – Part I	7-98
-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	Set terminal characteristics	7-45
4	Yes	Login	7-70
5	Yes	Logout	7-71
6	Yes	Attach	7-74
	Yes	Reattach	7-75
7	Yes	Detach	7-72
8	Yes	Change password/quota	7-49
	Yes	Kill job	7-51
	Yes	Disable terminal	7-52
9	No	Return error messages	7-28
10	No	Assign/reassign device	7-30
11	No	Deassign device	7-32
12	No	Deassign all devices	7-33
13	Both	Zero a device	7-34
14	Both	Read or Read and Reset Accounting Data	7-85
15	No	Directory lookup on index	7-90
	No	Special magtape directory lookup	7-92
16	Yes	Set terminal characteristics	7-63
17	No	Disk directory lookup on filename	7-94
	No	Disk wildcard directory lookup	7-96
18	No	Send a message	7-80
	Yes	Declaring a receiver and receiving a message	7-77
	Yes	Remove from receive table	7-82
19	Yes	Enable/disable disk caching	7-102
20	No	Reserved for special application	–
21	Yes	System logical names	7-113
22	Both	Message Send/Receive	Ch. 8
23	Yes	Add/Remove system files	7-117
24	–	Reserved for future use	–
–	Yes	PEEK function	7-122

[1] The privileged status column indicates whether the SYS call can be used only by a privileged user or by any user. A nonprivileged user who attempts to call a privileged SYS function always receives the ILLEGAL SYS ( ) USAGE error (ERR = 18). To avoid repetition in the documentation, error 18 is described for privileged calls only if it has a meaning different from nonprivileged attempts to use the call. The notation Both in the privileged status column indicates that some facilities of the specified function are available to a nonprivileged user, while the privileged user has a more powerful set.

[2] Poke memory can be executed only from account [1,1].

7.2.4.1 File Name String Scan – Not Privileged

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	The current job number times 2
2-4	Internal coding
5-6	Project-programmer number (0 means the current account)
7-10	File name in Radix-50 format; see discussion
11-12	Filename extension in Radix-50 format; see discussion
13-14	The number of blocks specified in the /FILESIZE:n (or /SIZE:n) file specification switch
15-16	The file cluster size given in the /CLUSTERSIZE:n file specification switch
17-18	The value for MODE specified in the /MODE:n (or /RONLY) file specification switch
19-20	Not used
21	If no protection code is found, this byte is 0 unless a default protection is currently assigned. If a protection code is found or if no protection code is found when a default protection is currently set, this byte is nonzero and byte 22 contains the protection.
22	Protection code when byte 21 is nonzero
23-24	To determine what is returned for a device, flag word 2 must be checked. If no colon was found in the string, these two bytes and byte 25 and 26 are 0. If a colon was found, a device name may or may not have been found. A device name can be a physical device name or a logical device name. If a physical device name was found, these bytes contain two characters in ASCII format. (For example, DK yields D in byte 23 and K in byte 24.) Bytes 25 and 26 contain unit number information. If a logical name (either job-specific

	or system-wide) was found and that logical name was translatable (the name was currently assigned to a physical device), the call translates the name and returns the full physical device information in bytes 23 through 26. If the logical device name was untranslatable (it was not currently assigned to a physical device), the call returns the logical name in Radix-50 format in bytes 23 through 26. Note that, if a physical device name is passed to this call and the device is not configured on the system, the name is treated as an untranslatable logical name.
25	If a physical device name is returned in bytes 23 and 24, this byte contains unit number information. The unit number here is real 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.
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. Refer to the *RSTS/E System User's Guide* for a description of a file specification. If the -10 version of the call is being used, the string may contain other than a valid file specification switch.	2
ILLEGAL NUMBER The argument on a file specification switch is missing, is greater than 65535 or contains an illegal character.	52
ILLEGAL SWITCH USAGE A file specification switch in the string scanned is not the last element in the file specification, is missing a colon, or is not a valid form of the switch.	67

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. The call also processes file specification switches allowed in RSTS/E. The format of these switches is described in the *RSTS/E System User's Guide.*

NOTE

This call is the only means provided to pack a string in Radix-50 format.

In particular, the call does the following for components of a file specification:

1. For a device specification, the call processes physical device names and unit number information. If a logical name is passed, the call attempts to translate it to a physical name. The STATUS variable is set for the device type found in the string scanned.

2. For a project-programmer specification, the call validates the format. If a character denoting an account is passed, the call translates it to the proper numbers. For example, $ is returned as 2 in byte 5 and 1 in byte 6. The call also indicates whether the wild card character was found.

3. For a file name, the call validates the format and translates names into Radix-50 format. It also notes the presence of wild card characters and detects file specification switches.

4. A file name extension is validated and translated into Radix-50 format. The call also notes the presence of wild card characters.

5. For a protection code, the call validates the format of the numbers. If a protection code is not found, the call returns the assigned value or, if an assignable code is not current, returns zero.

6. For file specification switches, the call validates the placement of the switches in the string and the format of each switch found. It notes the presence of those switches found and returns switch arguments.

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. Up to 9 characters can be packed in a single call if a period separates 6 characters and 3 characters (the file name and extension format).

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 information in them is accessible by means of an AND operation between the word and the bit relating to a particular piece of information. Each bit of the PDP-11 word can be used to hold a YES or NO answer.

Flag word 1 indicates whether file specification switches were detected in the string passed. Flag word 2 gives information concerning elements found in the file specification. The high byte of flag word 1 is retained for compatibility with previous versions of RSTS/E.

In Tables 7-3 and 7-4, 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.

Table 7-3 File Name String Scan Flag Word 1

Flag Word 1: where S0% = M%(27%)+SWAP%(M%(28%))		
Bit	**Comparison**	**Meaning**
0	(S0% AND 1%) < > 0% (S0% AND 1%) = 0%	The /CLUSTERSIZE:n switch was specified No /CLUSTERSIZE:n was found
1	(S0% AND 2%) < > 0% (S0% AND 2%) = 0%	Either the /MODE:n or /RONLY switch was specified Neither /MODE:n nor /RONLY was found
2	(S0% AND 4%) < > 0% (S0% AND 4%) = 0%	Either the /FILESIZE:n or /SIZE:n switch was specified Neither the /FILESIZE:n nor /SIZE:n switch was found
3-7		Not used
8	(S0% AND 256%) < > 0% (S0% AND 256%) = 0%	A filename was found in the source string (and is returned in Radix-50 format in bytes 7 through 10) No filename found
9	(S0% AND 512%) < > 0% (S0% AND 512%) = 0%	A dot was found in source string No dot was found in source string implying that no extension could have been specified either
10	(S0% AND 1024%) < > 0% (S0% AND 1024%) = 0%	A project-programmer number was found in source string No project-programmer number was found
11	(S0% AND 2048%) < > 0% (S0% AND 2048%) = 0%	A left angle bracket (<) was found in source string implying that a protection code was found No left angle bracket (<) was found (no protection was specified)
12	(S0% AND 4096%) < > 0% (S0% AND 4096%) = 0%	A colon (but not necessarily a device name) was found No colon was found implying that no device could have been specified
13	(S0% AND 8192%) < > 0% (S0% AND 8192%) = 0%	Device name was specified and specified device name was a logical device name 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 test bits of flag word 2 for wild card characters and device name found.

Table 7-4 File Name String Scan Flag Word 2

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[1]	(S1% AND 256%) < > 0%	Project number was an * character (that is the project-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[1]	(S1% AND 512%) < > 0%	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

[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 nonzero values.

(continued on next page)

Table 7-4 File Name String Scan Flag Word 2 (Cont.)

Bit	Comparison	Meaning
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; the following comparison returns 0
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.
	S1% > = 0%	The device name specified, if any, was either an actual device name, or a logical device name to which a physical device has been assigned. The physical device name is returned in bytes 23 and 24 and the unit information is returned in bytes 25 and 26.

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$,-1%)                              ! GET RID IF GARBAGE BYTES
40 CHANGE SYS(CHR$(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 FI 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:FILENM.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.

1. 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.
2. 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%).
3. 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 may be used in two versions. Both calls process RSTS/E file specification switches. The -10 version of the call processes a RSTS/E file specification only. If other than a valid form of a file specification switch is found, the ILLEGAL FILE NAME error is generated. The -23 version of the call processes a full command line which is allowed to contain multiple file specifications and switches other than valid forms of the file specification switches. To process a full command line, the call terminates the scan on certain characters.

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

= (equality sign)
/ (slant unless part of a valid file specification switch)
; (semi-colon)
, (comma unless between brackets or parentheses (ppn))
end of string

The scan is done from left to right. If a valid file specification switch is encountered, it is processed and the scan continues. If other than a file specification switch is found, the scan terminates. The program must process the switch and also check for remaining switches. Any file specification switches following a switch which terminates the scan are not processed.

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 (<) not part of a protection code, generate an error and none of the data is returned.

7.2.4.2 Return Error Message – Not Privileged

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	If attached, current keyboard number times 2 of terminal to which job is attached. If detached, the logical complement (NOT) of keyboard number times 2 from which job detached.
3-30	Error message. If message is less than 28 characters, remainder is 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 error message file installed during the current time sharing session or from the default error message file if an error message file is not currently installed. 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 on which the job is running. This is an exception to the conventional contents of byte 2 which usually contains internal data. 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%
        \S$=SYS(CHR$(6%)+CHR$(9%)+CHR$(E%))
        \S1$=CVT$$(RIGHT(S$,3%),4%)
        \PRINT S1$
        \PRINT FOR I%=1% TO 2%
        \GOTO 10
32767    END
```

To extract the message text from the data returned by the SYS call, the program executes a RIGHT( ) function to discard the first two bytes.The CVT$$( ) function discards any excess NUL characters. The first character of the text is the severity indication described in Appendix C.

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

```
RUNNH
ERROR NUMBER?  0
RSTS V06B-02 SYSTEM #880
```

7.2.4.3 Assign/Reassign Device – Not Privileged

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
9-10	Not used
11-12+	Either DOS or ANS in Radix-50 format to specify DOS or ANSI label format for the magtape drive.
13-22	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
For the assign call	
NOT A VALID DEVICE The device name specified in bytes 23 and 24 is a logical device name for which a physical device is currently not assigned.	6
DEVICE NOT AVAILABLE The device specified in bytes 23 through 26 exists on the system but the attempt to reserve it is prohibited for one of the following reasons. The device is currently reserved by another job. The device requires privilege for ownership and the user does not have privilege. The device or its controller has been disabled by the system manager. The device is a keyboard line for a pseudo keyboard use only.	8

Meaning	ERR Value
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 for which a physical device is currently not assigned.	6
DEVICE NOT AVAILABLE See the description above for the assign call.	8
ILLEGAL NUMBER An attempt is made to transfer control to a nonexistent job.	52

Discussion:

The assign call reserves a physical device to a job[1] or transfers assignment of a currently owned device to another job. The actions are equivalent to the ASSIGN dev: and REASSIGN dev:x monitor commands where x is the job number to which the device is reassigned. This call, however, can not make logical assignments. Job-specific logical assignments can be made only by keyboard command. System logical names are assigned either by the number 21 SYS call to FIP or by UTILTY program commands .

Example:

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

[1] The system manager, through an initialization option, can designate that certain devices require privilege to be assigned.

7.2.4.4 Deassign a Device – Not Privileged

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 or its type specified in bytes 23 through 26 is not configured on this system.	6

Discussion:

This call performs the same action as the DEASSIGN system command described in the *RSTS/E 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%) + STRING$(20%,0%) +
        "LP" + CHR$(1%) + CHR$(255%))
        ! DEASSIGN LP1:
```

7.2.4.5 Deassign All Devices – Not Privileged

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 meaningful data is 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

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). If no device is specified, SY: (the public structure) is used
25+	Unit number
26+	Unit number flag
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	**ERR Value**
ILLEGAL FILE NAME 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 or its type specified in bytes 23 through 26 is not configured on the system.	6
DEVICE NOT AVAILABLE The device specified in bytes 23 through 26 exists on the system but the attempt to zero it is prohibited for one of the following reasons. A file is currently open on the device. The device is currently reserved by another job. The device or its controller has been disabled by the system manager.	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 nonprivileged 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. Appendix A.3 describes what actions occur when magtape is zeroed.

Example:

```
10    A$=SYS(CHR$(6%)+CHR$(13%))
        ! ZERO MY OWN ACCOUNT ON THE SYSTEM.
20    P0%=10%
      \P1%=20%
        ! WANT TO ZERO [10,20]
30    A$=SYS(CHR$(6%)+CHR$(13%)+STRING$(2%,0%)+
       CHR$(P1%)+CHR$(P0%)+STRING$(24%,0%))
        ! ZERO [10,20] ON THE SYSTEM.
        ! IF PROGRAM IS NON-PRIVILEGED, ZEROES
        ! CURRENT ACCOUNT
40    A$=SYS(CHR$(6%)+CHR$(13%)+STRING$(20%,0%)+
        "MT"+CVT%$(0%))
        ! ZERO MT:
```

7.2.4.7 CTRL/C Trap Enable – Not Privileged

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 always possible to resume execution at the exact point where the CTRL/C occurred. The BASIC-PLUS variable LINE gives the number of the line being executed when the CTRL/C was typed. The variable ERL is not set when trapping is in effect and error 28 occurs. The variable ERL refers to the last error trapped by the program. The following sample routine shows the procedure.

```
  100 ON ERROR GOTO 1000
  200 X = X/0.0
      !THIS GIVES ERR 61, ERL 200
  300 Q$ = SYS(CHR$(6%) + CHR$(-7%))
      !SET CTRL/C TRAPPING
  400 SLEEP 100%
      \GOTO 400 .
      !WAIT FOR CTRL/C TO BE TYPED
 1000 RESUME 2000 IF ERR=28
      \RESUME 300 IF ERR=61
      \ON ERROR GOTO 0
 2000 PRINT LINE, ERL
32767 END
```

When the CTRL/C combination is typed at the terminal, the variable LINE is set to 400. The variable ERL remains set to 200 from error number 61 at line number 200.

Two methods are available to protect a program from CTRL/C aborts. One method is to open the console terminal in binary input mode described in Section 4.3.1 and detach the program. The second involves CTRL/C trapping with this SYS call. If certain critical sections of BASIC-PLUS code are to be protected with CTRL/C trapping, three actions must occur.

1. The job must detach itself from its terminal. See the description of FIP code +7.
2. The program must have CLOSEd all channels on which other terminals in the job had been OPENed.

3. 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.

1. The job can kill itself by means of FIP code +8.
2. 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.)
3. 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
        \A$ = SYS(CHR$(6%) + CHR$(-7%))
30      PRINT "HI ";
        \SLEEP 10%
        \GOTO 30

100     IF ERR <> 28% THEN ON ERROR GOTO 0
              ELSE RESUME 110
110     PRINT "CTRL/C TRAPPED"
        \SLEEP 10%
        \GOTO 10
32767   END
```

The program prints "HI" at the keyboard every ten seconds until a CTRL/C is typed. Then it prints the "CTRL/C TRAPPED" message and performs 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 descriptions. 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

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, bootstraps the initialization code 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).

7.2.5.2 Date and Time Changer – Privileged

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 *BASIC-PLUS Language Manual* 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 *BASIC-PLUS Langugage Manual* 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.

Possible Errors: No errors are possible.

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

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. If no value is specified, the line is hung up after 2 seconds.
5-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors: No errors are possible.

Discussion:

This SYS call allows a dial up line to be disconnected under program control. A dial up line can be connected but not be performing any processing. This condition prevents other users from gaining access to the system.

7.2.5.4 Broadcast to a Terminal – Privileged

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.

7.2.5.5 Force Input to a Terminal – Privileged

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.

7.2.5.6 Disable Further Logins – Privileged

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:

Byte(s)	Meaning
1	The current job number times 2
2	Not used
3	CHR$(N%) where N% is the number of jobs currently logged into the system
4-30	Not used

Possible Errors: No errors are possible.

Discussion:

This call sets the number of logins allowed on the system to 1.[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.

[1]The system manager can install a patch which enables one specific terminal on the system to login even if the new job exceeds the currently set maximum number of logins. On all systems, a user can log into the system on KB0: regardless of the number of logins allowed.

7.2.5.7 Enable Further Logins – Privileged

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:

Byte(s)	Meaning
1	CHR$(J%) where J% is the job number times 2 of the job executing this call
2	Not used
3	CHR$(N%) where N% is the number of logins allowed
4-30	Not used

Possible Errors: No errors are possible.

Discussion:

This call sets the number of logins allowed to the maximum number possible, given that swapping file space may have been added. The number will never exceed that specified at start-up time – JOBMAX.

7.2.5.8 Disk Pack and Terminal Status – Privileged

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
11-12	CHR$(F%) where F% is a flag which determines whether a logical name is to be used. If both bytes are 0, the call attempts to use the pack identification. If both bytes are 255%, the call attempts to substitute the name given in bytes 13 through 16 for the pack identification.
13-16	Logical name for this disk, in Radix-50 format. If bytes 11-12 are 255%, the logical name given here supplants the pack identification as the system-wide logical name. If bytes 11-12 are 255% and these bytes are 0, the system places 0 in the logical name table.
17-18	Mode word. If the sign bit is not set (bit 15 is 0), a mode value is not used on the mount operation. If the sign bit is set (32767%+1% is included in the value of this word), the following bit definitions are recognized. 16384% for private only usage 8192% for read only access

Data Returned: Data is returned on mount only.

Byte(s)	**Meaning**
1	The current job number times 2
2-12	Not used
13-16	Logical name used for this disk, in Radix-50 format
17-30	Not used

Possible Errors:

Meaning	**ERR Value**
ACCOUNT OR DEVICE IN USE An attempt is made to dismount a disk which has an open file.	3
NOT A VALID DEVICE The device specification supplied in bytes 23 through 26 is illegal because the unit or its type is not configured on the system.	6
DEVICE HUNG OR WRITE LOCKED An attempt is made to mount a disk which is not write enabled.	14
ILLEGAL SYS( ) USAGE An attempt to mount a disk which is already mounted or which resides in a non-dismounted drive; or disk specified is the system disk.	18
PACK IDS DON'T MATCH An attempt is made 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 because the disk was not properly dismounted when it was last used. Disk is logically mounted but should not be accessed until the storage allocation table is rebuilt by the CLEAN command of UTILTY system program or by the FIP call +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 is made to lock, unlock, or dismount a disk currently opened in non-file structured mode.	30

Discussion:

Note that if byte 3 contains any 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. (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.*

The mode value given in bytes 17 and 18 of the mount call allows the mounting operation to be modified. These bits are used by the MOUNT command to enable /RONLY and /PRIVATE operations.

The mount version of this call first mounts the disk pack or cartridge and then determines whether a logical name should be placed in the system logical name table. If the mount operation fails, an error is returned to the program. If the mount succeeds, the call checks bytes 11 and 12 of the data passed.

NUL characters in bytes 11 and 12 mean that the pack identification is to be placed in the table as the logical name for that disk unit. The call scans the entire table. If the name is not currently in use, the pack identification is placed in the table and is written in bytes 13 through 16 of the data returned to the program. This action notifies the program that a logical name is current for that disk unit. If the pack identification is currently in use as a logical name for another device, the call writes NUL bytes in the table. To notify the program that a logical name was not placed in the table, NUL characters are written in bytes 13 through 16 of the data returned. No error is returned to the program because the mount operation itself succeeded.

If bytes 11 and 12 of the data passed are 255%, the call attempts to place in the logical name table the name found in bytes 13 through 16. If bytes 13 through 16 contain NUL bytes, no name is placed in the table. When bytes 13 through 16 contain a logical name, the call performs the same actions as described above for the pack identification to place the name in the table. The program should check the data returned to determine whether a logical name is in effect. If the call found the logical name currently in use, it does not attempt to use the pack identification.

7.2.5.9 Clean Up a Disk Pack – Privileged

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 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 should be done whenever the DISK PACK NEEDS 'CLEANING' error (ERR=25) occurs on a mount operation. The system sets a bit on a disk when it is mounted. The bit is cleared by the dismount code. If the system finds the bit set upon mounting the disk, chances are the disk was not properly dismounted and the storage allocation tables may be incorrect. The system generates the warning error. The clean operation thus checks through all directories on the disk and ensures that the SATT.SYS file reflects current allocation. This action essentially rebuilds the storage allocation table. 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

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 means 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$(255%) if the quota of 0 specified in bytes 13 and 14 is valid rather than no change.
23-24+	Device name. If no device name is specified, SY: is used.
25+	Unit number
26+	Unit number flag
27	Not used
28	Must be CHR$(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 The device specification supplied in bytes 23 through 26 is illegal because the unit or its type is not configured on the system.	6
ILLEGAL SYS( ) USAGE 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

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.	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.

7.2.5.12 Disable Terminal – Privileged

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; keyboard number corresponds to a line used by a pseudo keyboard; keyboard number relates to a terminal on a multiplexer line; or the terminal is currently opened or assigned by a job.	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 sharing session is started.

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

This call cannot disable the system console terminal (KB0:). Disabling KB0: is a dangerous operation because the SHUTUP system program runs only on that terminal.

To disable a terminal (other than KB0:) during multiple time sharing sessions, it is recommended that the SET option be executed as described in the *RSTS/E System Generation Manual.*

To disable a terminal connected to a multiplexer line, set the line's input speed to 0 (via the set terminal characteristics SYS call F0=16), force a CTRL/C combination on the line (via the force input to a terminal SYS call F0=4), and set the output speed to 0 (via the set terminal characteristics SYS call F0=16).

With the input speed of 0, input is disabled. Forcing the CTRL/C combination clears all pending buffers. The output speed of 0 disables output.

7.2.5.13 Add/Delete CCL Command – Privileged

Data Passed:

To add a CCL command, specify the bytes described below.

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-24%), the add/delete CCL code
3	CHR$(0%) to add a CCL command
4	CHR$(U%) where U% is the number of unique characters in the command. U% must be between 1 and the length of the command. This defines the abbreviation point.
5-6	Project-programmer number under which program to run is stored
7-10	Filename, in Radix-50 format, of the program to run
11-12	Filename extension, in Radix-50 format, of the program to run
13-21	CCL command; from 1 to 9 ASCII characters padded with NUL characters
22	Must be CHR$(0%)
23-24	Name of device on which program to run is stored (must be disk)
25	Device unit number if byte 26 is 255
26	If this byte is 255, the value specified in byte 25 is the explicitly specified unit number.
27-28	Line number at which to start program (add 32767% + 1% to keep privileges)
29-30	Not used

To delete a CCL command, specify the bytes described below.

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-24%), the add/delete CCL code
3	CHR$(2%) to delete a CCL command
4-12	Unused
13-21	CCL command to delete
22-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
For the add CCL call	
ILLEGAL FILE NAME The CCL command being added either begins with a number or contains an otherwise unacceptable character.	2
ACCOUNT OR DEVICE IN USE The CCL command being added is already defined.	3
For the delete CCL call	
CAN'T FIND FILE OR ACCOUNT The CCL command specified does not exist	5

Discussion:

Concise command language commands can be added and deleted by this call. Section 9.2 of this manual presents the operation and design of CCL commands.

The command can be from 1 to 9 characters long. The allowable characters are A through Z, inclusive, 0 through 9 inclusive, and the commercial at (@) character. The command may not begin with a numeric character because BASIC-PLUS interprets line numbered input in a special manner.

Commands have an abbreviation point which is after the first character. The abbreviation point is specified by the value in byte 4. If the abbreviation point follows the last character of the command, the command can not be abbreviated. For example, DIR (the abbreviation point follows the R) can uniquely define the CCL command DIRECTORY. Any of the following abbreviations are valid: DIR, DIRE, DIREC, DIRECT, DIRECTO, DIRECTOR, and DIRECTORY. If the abbreviation point for DIRECTORY follows the Y, then no abbreviation is valid.

7.2.6 Job Scheduling SYS Calls to FIP

7.2.6.1 Priority, Run Burst and Size Maximum Changer – Privileged

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$(J%), where J% is the job number affected or is 255% to denote the current running job
4	CHR$(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$(P%) where P% is the value of the running priority and ranges from -128 to +120 in steps of 8.
6	CHR$(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 1 and 255. If this value exceeds SWAP MAX, the value of SWAP MAX is used by the system.

Data Returned: No meaningful data is returned.

Possible Errors: No errors are possible.

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 initial size of a job running under the BASIC-PLUS run-time system 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 the *RSTS/E System Generation Manual*). The maximum size to which a job can grow can never be greater than the currently assigned value of SWAP MAX, which should be between 1K and 28K words (8K and 16K words for BASIC-PLUS jobs). Therefore, the privileged user has the option of limiting the size to which a BASIC-PLUS 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.

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 priority to -128 suspends that job. The monitor does not schedule that job to run again until its priority is set to a value other than -128. 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

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: No errors are possible.

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 or whenever a program CHAINs to another program. Thus, a privileged job can raise its priority without protecting against a user typing CTRL/C and retaining the higher priority.

7.2.6.3 Lock/Unlock Job in Memory – Privileged

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: No errors are possible.

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 cause fragmentation of user space and 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/Regain Temporary Privileges – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-21%), the drop temporary privileges code
3	If unspecified, the call permanently drops temporary privileges. Otherwise, CHR$(N%) where N% means the following: 255% temporarily drop temporary privileges 0% regain temporary privileges dropped by 255% value above.
4-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors: No errors are possible.

Discussion:

The call allows a program to either temporarily or permanently drop its temporary privilege. (Privilege is defined in Chapter 1.) If a program temporarily drops its temporary privilege, it can use this call to regain the privilege.

A program normally executes this call after it has used the temporary 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.

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

```
10   A$ = SYS(CHR$(6%) + CHR$(-22%))
     ! SET SPECIAL RUN PRIORITY BEFORE DROPPING TEMP PRIV'S

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

30   A$ = SYS(CHR$(6%) + CHR$(-21%) + CHR$(-1%))

     ! TEMPORARILY DROP TEMPORARY PRIVILEGES

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

50   A$ = SYS(CHR$(6%) + CHR$(-21%) + CHR$(0%))
     ! REGAIN PRIVILEGES IF OPEN IS SUCCESSFUL
```

7.2.7 Account Creation and Deletion SYS Functions

7.2.7.1 Create User Account – Privileged

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 as an unsigned number. See Section 7.2.3 for a description of unsigned numbers. 0 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

Meaning	ERR Value
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 Chapter 1 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 creates accounts on a private disk or on the public structure.

7.2.7.2 Delete User Account – Privileged

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, for an account being deleted from the public structure, 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:

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

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. (The 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$(N%) where N% is: 0 for no change 128 to enable the software to perform form feed and vertical tab operations by executing four line feed operations. The NO FORM command sets this characteristic. 255 to enable hardware form feed and vertical tab. The FORM command sets this characteristic. (The device must have the requisite hardware.)
8	CHR$(N%) where N% is: 0 for no change 128 to allow terminal to receive and print lower case characters (CHR$(96%) through CHR$(126%)). The LC OUTPUT command sets this characteristic. 255 to have the system translate lower case characters to upper case before transmitting to a terminal. The NO LOC 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

Byte(s)	**Meaning**
	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 set this characteristic. 255 Terminal is a video display, or cathode ray tube (CRT) type, and uses the following features: a. Responds to synchronization protocol 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) (unless the ESC characteristic is subsequently set). b. Translate lower case characters (CHR$(64%) through CHR$(94%)) to upper case equivalents (CHR$(96%) through CHR$(126%)). 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.

Byte(s) **Meaning**

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 *PDP-11 Peripherals Handbook* 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 Generate 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 protocol described for 255% value. The NO STALL command sets this characteristic.
- 255 Terminal obeys the synchronization protocol 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 No effect.

Byte(s)	Meaning
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$(27) character and the following incoming characters as a special escape sequence. Refer to Section 4.4.2 for a description of incoming escape sequences. The ESC SEQ command sets this value.
23	CHR$(N%) where N% is: 0 for no change 128 Disable (clear) the private delimiter. The DELIMITER command sets this value. 128+n Set the private delimiter to ASCII code n (between 1 and 127). If the character has special meaning (for example, horizontal tab or the CTRL/Z combination), the private delimiter usage has higher precedence. The delimiter can not be used with INPUT, INPUT LINE, and MAT INPUT statements. See Section 4.1 for a discussion of delimiters.

Byte(s)	Meaning
	The DELIMITER x and DELIMITER 'x' commands set this value.
24	CHR$(N%) where N% is: 0 for no change 128 The terminal being used does not have the ESC key which generates CHR$(27%). Therefore, translate ALT MODE CHR$(125%) and PREFIX CHR$(126%) to CHR$(27%). The NO ESC command sets this value. 255 The terminal being used has an ESC key which generates CHR$(27%). Therefore, do not translate CHR$(125%) and CHR$(126%) but treat them as their ASCII characters } and ~. The ESC 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 the *RSTS/E System User's Guide.*

7.2.9 **Change File Statistics** – Privileged

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[1]
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.	18

Discussion:

The data passed by this call replaces the related data in the accounting entry 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 following is a partial directory listing of a privileged account showing the file whose statistic information is to be changed.

```
CAT
CTPBLD.BAS          0        60       30-Sep-76 30-Sep-76 03:13 PM

Ready
```

[1]The DSKINT initialization option and system program can change the meaning of date of last access to date of last modification. The disk status report of SYSTAT and the display program tells which disks record date of last modification.

The following program changes the date and time of creation to 12:00 noon, 21-Jul-76, and the date of last access to 21-Jul-76, as shown on the partial directory listing following the program.

```
LISTNH
10      D% = 6203%
        !21-JUL-76 IS (203) + ((1976-1970)*1000)
20      T% = (24% * 60%) - (12 * 60)
        !12 NOON IS 720 MINUTES BEFORE MIDNIGHT
100     OPEN 'CTPBLD.BAS' AS FILE 1%
        \DIM M%(30%)
        \M%(0%) = 30%
        \M%(1%) = 6%
        !OPEN FILE TO CHANGE, USE ARRAY TO SET UP CALL
200     M%(2%) = -11%
        \M%(3%) = 1%
        !SET UP FOR CHANGE STATS CALL ON CHANNEL 1
300     M%(4%) = D% AND 255%
        \M%(5%) = SWAP%(D%) AND 255%
        !SET UP THE DATE OF CREATION
400     M%(6%) = D% AND 255%
        \M%(7%) = SWAP%(D%) AND 255%
        !SET UP THE DATE OF LAST ACCESS
500     M%(8%) = T% AND 255%
        \M%(9%) = SWAP%(T%) AND 255%
        !SET TIME OF CREATION TO T%
1000    CHANGE M% TO M$
        \M$ = SYS(M$)
        !SET ARRAY UP AS STRING AND DO CALL
32767   END

Ready

RUNNH

Ready

CAT
CTPBLD.BAS      0     60     21-Jul-76 21-Jul-76 12:00 M
TEMP20.TMP      1     60     30-Sep-76 30-Sep-76 03:14 PM

Ready
```

7.2.10 LOGIN and LOGOUT SYS Calls

7.2.10.1 LOGIN – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(4%), the LOGIN code
3-4	Not used
5-6+	Project-programmer number. Must not be account [0,1].
7-10+	Password in Radix-50 format
11-30	Not used

Data Returned:

Byte(s)	Meaning
1	The current job number times 2
2	Not used
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 not 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

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: No errors are possible.

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. 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

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(7%), the detach code
3	CHR$(J%+C%) where J% is the number of the job to detach and C% is the CLOSE flag. If J% is 0% or byte 3 is not specified, the calling job is detached. J% can be between 1% and the current job maximum to detach another job. If C% is 0% or byte 3 is not specified, the system does not force a CLOSE on the terminal. If C% is 128%, the system forces a CLOSE on all channels on which the terminal is open after detaching the job.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL SYS( ) USAGE The current job is already detached.	18

Discussion:

This call disassociates the calling job or another 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 free 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 the *RSTS/E 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.

The system associates I/O channel 0 with the terminal which activated the job. The terminal associated with channel 0 is called the job's console terminal or console keyboard.

By executing this call, a privileged job can become detached from its console terminal or can detach another job from that job's console terminal. 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 the *RSTS/E 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 is 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.

If the detached job has its console terminal open on some nonzero channel, it can perform I/O on the keyboard from which it is detached. It must use a nonzero I/O channel. An attempt by a detached job to perform I/O on channel 0 causes the system to place it in the hibernate state. The I/O TO DETACHED KEYBOARD (ERR=27) is generated. A detached job which performs I/O on the detached keyboard on a nonzero channel, however, retains control of the terminal. The terminal, therefore, is not free for other use.

If 128% is not specified in byte 3 of the data passed, the system disassociates the job and its console terminal but does not change the status of the keyboard on any nonzero I/O channel. The detached job can continue to perform I/O on the nonzero channel. Channel 0 I/O still causes hibernation. In this circumstance, the job is detached but the console terminal remains in use by the job and is not free for other use.

By specifying 128% in byte 3 of the data passed, the program forces the system to close the terminal on any nonzero I/O channel being used. The disassociation which the detach call performs then includes all channels on which the console terminal is open. The keyboard from which the job is detached is explicitly forced to be free. An attempt by the detached job to perform I/O on the terminal on the nonzero channel causes the system to place the job in the hibernate state.

7.2.11.2 ·**Attach** – Privileged

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 format of the reattach SYS call to FIP
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
7-10+	Password of the account specified in bytes 5 and 6 in 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: 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.	18

Discussion:

The LOGIN system program executes 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

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: 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.	18

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

NOTE
This call is documented for compatibility with previous implementations of message processing. For a complete description of current message processing, refer to Chapter 8.

This function call allows a user's job to do the following:

1. declare itself a receiving job,
2. send a message to a receiving job,
3. receive a message from pending messages,
4. stall until a message is pending, and
5. eliminate itself of another job as a receiving job.

The monitor controls eligibility to receive messages by maintaining a table of receiving jobs. 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

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$(N%) where N% is one of the following values: 1. Attempt to receive a message or declare this job as a receiver and attempt to receive a message. Return an error condition if no messages are pending. 2. Receive with sleep. Similar to 1 except that the job executes a sleep operation if no messages are pending. This action occurs in place of generating an error condition immediately. When a message is pending, the system awakens the job and returns an error condition.
4	CHR$(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	The current job number times 2
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

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.
29-30	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 BUFFER SPACE AVAILABLE 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

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: No errors are possible.

Discussion:

This function removes the job number and logical name from the receive table. All pending messages are lost.

7.2.13 Poke Memory – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-6%), the poke core code
3-4	CHR$(A%)+CHR$(SWAP%(A%)) where A% is the address to change
5-6	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

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:

Byte(s)	Meaning
1	The current job number times 2
2	Not used
3	CHR$(N%) where N% is the actual number of logins set
4-30	Not used

Possible Errors: No errors are possible.

Discussion:

This function sets the number of allowable logins to the number specified in byte 3.[1] 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.

[1]The system manager can install a patch which enables one specific terminal on the system to login even if the new job exceeds the currently set maximum number of logins. On all systems, if a swapping space is available, a user can log into the system on KB0: regardless of the number of logins allowed.

7.2.15 Accounting Information

7.2.15.1 Read or Read and Reset Accounting Data – Both

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. If bytes 7 and 8 are 0, data for the current account are returned. 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 SY: (the public structure).
25+	Unit number
26+	Unit number flag
27-30	Not used

Data Returned:

Byte(s)	Meaning
1	The current job number times 2
2	Not used
3-4	Same as bytes 3-4 in Data Passed.
5-6	Number of blocks owned by the account read
7-8	Project-programmer number of the account read. This data is returned only if caller is privileged
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

Byte(s)	Meaning
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	Same as bytes 23-26 in Data Passed
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 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 as shown in Figure 7-3.

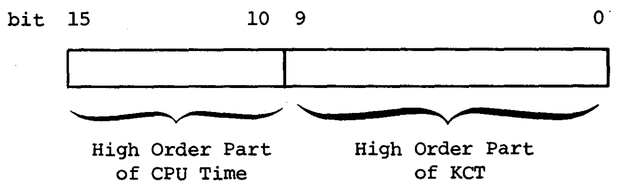


Figure 7-3 High Order Bits of CPU Time and KCTs.

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

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-15%), the accounting dump code
3-4	Not used
5-6+	Project-programmer number of the account to which the system dumps the accumulated usage data If both bytes are zero, the data is dumped to the current account.
7-30	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 5 and 6 does not exist.	5

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 nonprivileged 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 takes advantage of this SYS call to effect this process.

This call forces the accumulated accounting values in memory to be written to disk. The values in memory are zeroed. To charge accounting data to another user's account, perform these steps.

1. Dump accounting data to the current account. This zeroes the data.
2. Perform processing for the account to be charged.
3. Dump accounting data to the account to be charged.

This procedure ensures that only the time expended for another account is charged to that account.

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	The current job number times 2
2	Not used
3-4	Same as bytes 3-4 in Data Passed.
5-6	Project-programmer number (if applicable) of the file read.
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. (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;[1] for tape, not used
19-20	The date of creation
21-22	The time of creation
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	The USTAT byte from the UFD Name entry

This byte contains internal flag information encoded as follows:

Bit Value	**Meaning**
1	Retained for historical purposes
2	Reserved for future use
4	Some job has write access now
8	File is open in update mode
16	File is contiguous; no extend available
32	No delete or rename allowed
64	MFD type entry
128	File is marked for deletion

[1]The system manager can use the DSKINT initialization option on a particular disk to change the meaning of date of last access to date of last modification.

7.2.16.1 Directory Look Up on Index – Not Privileged

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 disk, the call returns information for the current account. 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. If both bytes are 0, SY: (the public structure) is used
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

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. 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 or MM
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:

1. Reads one record from the tape (a label record).[1]
2. Spaces the tape forward to the next end of file record and calculates the number of records in the file.
3. 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.

[1]The procedure works for either DOS or ANSI labeling. The description, however, does not distinguish between the different types of label records in ANSI processing.

When the monitor executes the action described in statement 1, 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.

1. Reads one record from the tape which must be a label record.
2. If the read operation is successful, then opens the file and returns control to the user program.
3. If the read operation is unsuccessful and this is the first label read, then rewinds the tape and executes the action described at 1.
4. If the logical end of tape is detected, returns an error.
5. If the label read does not match, skips to the end of this file and executes the action at 1.

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.

1. Reads from the tape a record which must be a label record.
2. Backspaces one record which leaves the tape in a position to read the label record again.
3. 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.

1. Determine from the information returned whether the file is the one required.
2. 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 2.8.4).
3. 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

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. If both bytes are 0, the current account is used.
7-10+	File name in Radix-50 format
11-12	Extension in Radix-50 format
13-22	Not used
23-24+	Device name; must be disk. If both bytes are 0, SY: (the public structure) is used
25+	Unit number
26+	Unit number flag
27-30	Not used

Data Returned: See introductory material for Section 7.2.16.

The following bytes differ for this call.

Byte(s)	Meaning
23-26	If a logical device name (for example, SY:) is passed, the real device designation (for example, DB2:) is returned
29-30	For RSXLIB utilities, the RSX-11M FID (file identification) is returned

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

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.
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.

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 CNT.KB in byte 3. The value returned is the number of terminal lines minus 1 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

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	The current job number times 2
2	Not used
3	CNT.KB-1 - the maximum keyboard number 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	(MEMLST) - the root link word in the first memory control subblock
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 (low byte) and error counts (high byte) for disks
19-20	(UNTCNT) - the status table of all disk 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	(JSBTBL) - the table of job status bits ordered by driver index

Byte(s)	**Meaning**
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

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	The current job number times 2
2	Not used
3-4	(FREES) - the table of free (small and large) buffer information
5-6	(DEVNAM) - the device name table
7-8	(CSRTBL) - the CSR table
9-10	(DEVOKB) - the number of disk devices times 2 in the DEVNAM table
11-12	(TTYHCT) - the number of hung terminal errors since system start up
13-14	(JOBCNT) - the count of jobs currently running (low byte) and the number of logins currently allowed (high byte)
15-16	(RTSLST) - the root link word in the linked list of Run Time System description blocks
17-18	(ERLCTL) - error logging control data
19-20	(SNDLST) - the list of eligible message receiving jobs
21-22	(LOGNAM) - the table of system logical names
23-24	(DEVSYN) - start of synonym names in DEVNAM
25-26	(MEMSIZ) - 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. Size is in K words times 32.
27-28	(CCLLST) - the root link word in the linked list of concise command language description blocks
29-30	Not used

Possible Errors: No errors are possible.

7.2.17.3 Get Open Channel Statistics – Not Privileged

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	The current job number times 2
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 Caching – Privileged

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: No errors are possible.

Discussion:

This function enables or disables the FIP buffering module. The ENABLE CACHE and DISABLE CACHE commands of the UTILTY program use this call.

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. These calls are used by the UTILTY system program described in the *RSTS/E System Manager's Guide.*

7.2.19.1 Name a Run-Time System – Not Privileged

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 Radix-50 format
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
PROTECTION VIOLATION The file open on the channel specified in byte 3 is not a disk file or the job executing the call does not have write access to the file.	10

Discussion:

This SYS function writes the name of the run-time system given in bytes 4 through 7 to the file open on the channel specified in byte 3.

Every file on RSTS/E has an associated run-time system under which it was created. The name of the run-time system is stored in Radix-50 format in the file's UFD accounting entry. The monitor looks at this run-time system name only for executable files on RUN requests. This call is used by utility programs to allow an executable file created by another run-time system to be run under an auxiliary run-time system supported in RSTS/E.

7.2.19.2 Add a Run-Time System – Privileged

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 Radix-50 format
11-12	CHR$(A%) where A% is the 1K-word section of memory at which this run-time system is to be loaded. The numbering begins at 0 and ends at n–1 (where n is the total number of 1K-word sections of memory on the system). If A% is 0%, the monitor decides where to load the run-time system.
13-14	Maximum allowed user image size, in K words (the P.SIZE symbol)
15-16	Minimum allowed user image size, in K words (the P.MSIZ symbol)
17	CHR$(P%) where P% is the position in the linked list of run-time system description blocks to place the description block for this run-time system. If P% is 1%, the description block is placed immediately after that of the system default RTS. If P% is a nonzero value less than or equal to the number of blocks currently in the list, this new block is placed in that position following the system default RTS block. If P% is 0% or a value greater than the number of blocks currently in the list, this new block is placed at the end of the list.
18	CHR$(S%) where S% is the stay flag. If S% is 128% (the high bit is set), this RTS is kept permanently resident. (The usage count remains non-zero.) If S% is 0%, the memory occupied by this RTS can be released as user job space whenever the usage count of the RTS goes to 0.
19-20	CHR$(F%)+CHR$(SWAP%(F%)) where F% is a flag word whose bits define this run-time system's characteristics. Only the high byte is used for flag bits. F% is the sum of the bits set as follows.

256%	This RTS is a keyboard monitor
512%	This RTS handles only one user (that is, it is not shared by multiple users)
1024%	This RTS allows read and write access to its memory rather than read only access
2048%	This RTS does not want errors occurring under its control to be recorded in the system error log
4096%	This RTS should be immediately removed from memory when its usage count goes to 0
8192%	The proper job image size (in K-words) is computed as (file-size+3)/4
16384%	Not used
32767%+1%	This RTS emulates trap instructions by using a special EMT prefix. If this characteristic is specified, the EMT prefix code is in the low byte ($0 \leqslant$ code $\leqslant 255$)

Byte(s)	**Meaning**
21-22	The normal executable file extension, in Radix-50 format, for this run time system (the P.DEXT symbol).
23-24	Name of the device (must be disk) on which the run time system file is stored. If no name is specified, SY: is used.
25	Unit number
26	Unit number flag
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	**ERR Value**
NO ROOM FOR USER ON DEVICE If the monitor were to load this run-time system at the address specified in bytes 11 and 12, memory would be fragmented and a swapping violation would occur. Refer to the discussion of assigning and allocating memory in the *RSTS/E System Generation Manual* for guidelines on how to avoid fragmenting memory.	4
CAN'T FIND FILE OR ACCOUNT A file with the name specified in bytes 7 through 10 and with an extension of .RTS cannot be found in account [0,1] on the device given in bytes 23 through 26	5
PROTECTION VIOLATION The file to be added as the run-time system has a bad format. For example, the file is not contiguous or has illegal entries in the SIL index.	10
NAME OR ACCOUNT NOW EXISTS A run-time system with the same name currently exists.	16
ILLEGAL BYTE COUNT FOR I/O The range of memory starting at the load address given in bytes 11 and 12 is not available. Refer to the memory status report of a display program to select an available range of memory.	31
NO BUFFER SPACE AVAILABLE Adding a run-time system description block requires a small buffer and one is not currently available.	32

Discussion:

This SYS function adds a run-time system description block to the linked list of blocks in the monitor. Run-time systems other than the system default run-time system are transient from one time sharing session to another. Because of this transiency, systems which offer auxiliary run-time systems must define them for each time sharing session.

7.2.19.3 Remove a Run-Time System – Privileged

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 Radix-50 format
11-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ACCOUNT OR DEVICE IN USE This run-time system is currently being loaded into memory or is resident and in use. It cannot be removed until usage count is 0.	3
CAN'T FIND FILE OR ACCOUNT The run-time system specified in bytes 7 through 10 is not currently defined.	5
PROTECTION VIOLATION The run-time system specified in bytes 7 through 10 is the system default RTS and cannot be removed by this call. Use the DEFAULT initialization option to change system default RTS before starting time sharing.	10

Discussion:

The SHUTUP system program automatically performs the remove action when time sharing operations are terminated. The monitor structure which defines this run-time system is deleted and the run-time system file is closed.

7.2.19.4 Load a Run-Time System – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-18%), the run-time system manipulation code
3	CHR$(2%), load run-time system
4-6	Not used
7-10+	Run-time system name in Radix-50 format
11-12	CHR$(A%) where A% is the 1K-word section of memory at which this run-time system is to be loaded. The numbering begins at 0 and ends at n-1 (where n is the total number of 1K-word sections of memory on the system). If A% is 0, the load address is determined by the address given when the RTS was added. If the add load address was zero also, then the monitor determines where to load the RTS. If the add load address was nonzero, the monitor uses that as the load address. If A% is nonzero and the add load address was 0, the monitor uses A% for the first residency and for later residencies decides the load address. If A% is nonzero and the add load address was also nonzero, the monitor uses A% for first and later residencies.
13-17	Not used
18	CHR$(S%) where S% is the stay flag. If S% is 128% (the high bit is set), this RTS is kept permanently resident. (The usage count remains non-zero.) If S% is 0%, the memory occupied by this RTS can be released as user job space whenever the usage count of the RTS goes to 0.
19-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
NO ROOM FOR USER ON DEVICE If the monitor were to load this run-time system at the address specified in bytes 11 and 12, memory would be fragmented and a swapping violation would occur.	4
CAN'T FIND FILE OR ACCOUNT The run-time system specified in bytes 7 through 10 is not currently defined.	5

Meaning	**ERR Value**
PROTECTION VIOLATION The run-time system specified in bytes 7 through 10 is the system default RTS and has a preset loading address. Use the DEFAULT initialization option to relocate the system default RTS.	10
ILLEGAL BYTE COUNT FOR I/O The range of memory starting at the load address given in bytes 11 and 12 is not available. Refer to the memory status report of a display program to select an available range of memory.	31

Discussion:

This SYS call loads the run-time system described in bytes 7 through 10. It is useful when a run-time system, currently positioned at a consistent location in memory by an add load address, must be moved to another part of memory. The LOAD command of the UTILTY program performs this function. If a run-time system, currently not positioned at any certain location in memory, must be moved permanently to another location, the stay bit (specified in byte 18) can be set to keep it resident during the current time sharing session.

7.2.19.5 Unload a Run-Time System – Privileged

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 Radix-50 format
11-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ACCOUNT OR DEVICE IN USE The run-time system specified in bytes 7 through 10 is currently being loaded into memory or is resident and in use. It cannot be unloaded until usage count is 0.	3
CAN'T FIND FILE OR ACCOUNT The run-time system specified in bytes 7 through 10 is not currently defined.	5
PROTECTION VIOLATION The run-time system specified in bytes 7 through 10 is the system default RTS and must remain resident during time sharing.	10

Discussion:

This call frees the position of memory occupied by the run-time system. The memory is made available as user job space.

7.2.20 Read and Write Attributes

Certain PDP-11 record organizations define characteristics for files which they create. These characteristics are called file attributes. The attributes of a file are defined when the file is created and must be retained during the existence of the file.

In RSTS/E, attributes are retained on disk in a UFD entry.

7.2.20.1 Read Attributes – Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-25%), the read and write attributes code
3	CHR$(N%), where N% is the channel number on which the file is open
4	CHR$(0%) to specify read
5-30	Not used

Data Returned:

Byte(s)	Meaning
1-4	Not used
5-26	File attribute data. If file has no attributes, bytes 5 and 6 contain zeroes.
27-30	Name of run time system, under which file was created, in Radix-50 format.

Possible Errors:

Meaning	ERR Value
I/O CHANNEL NOT OPEN Channel specified in byte 3 must have file open.	9
PROTECTION VIOLATION Job does not have read access to the file.	10
DEVICE NOT FILE STRUCTURED Device on which file is open must be disk.	30
ILLEGAL I/O CHANNEL Attributes can be written only on channels 1 through 12.	46

7.2.20.2 Write Attributes – Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(-25%), the read and write attributes code
3	CHR$(N%), where N% is the channel number on which file is open
4	CHR$(N%), where N% is the number of words to write. (1<N<11)
5-26	The attribute data to write, 2 bytes per attribute
27-30	Not used

Data Returned: No data is returned.

Possible Errors:

Meaning	ERR Value
NO ROOM FOR USER ON DEVICE The UFD of the account is full. Some files must be deleted to free entries for attributes.	4
I/O CHANNEL NOT OPEN Channel specified in byte 3 must have file open.	9
PROTECTION VIOLATION Job does not have write access to the file open on channel.	10
DEVICE NOT FILE STRUCTURED Device on which file is open must be disk.	30
ILLEGAL BYTE COUNT FOR I/O No more than 11 can be specified in byte 4.	31
ILLEGAL I/O CHANNEL Attributes can be written only on channels 1 through 12.	46

7.2.21 System Logical Names – Privileged

RSTS/E allows users to access devices by logical names as well as by physical names. Logical names which apply to all users are termed system logical names. On all systems, users can refer to a disk by its pack identification or a name which replaces the pack identification. Similarly, the capability exists to define logical names for nondisk devices and additional logical names for disk devices.

This SYS call allows a programmer to add, remove and change system logical names. The total number of additional system logical names allowed is a system generation parameter and varies from system to system.[1] The UTILTY system program uses this call to add, remove, and change system logical names for the system manager.

RSTS/E maintains a table of system logical names in two parts. The first part exists on all systems and contains an entry for each disk unit configured on the system. The position of an entry is fixed to a specific disk type and unit and never has a project-programmer number associated with the logical name.

The second part of the logical name table is optional and exists only if space for entries was configured. The position of entries in the second part is dynamic. Multiple entries are allowed for a specific device and unit. Only one entry, however, can appear for any specific logical name. Additionally, an entry in the second part of the table can have a project-programmer number associated with the logical name. This mechanism allows a default account specification to be applied for a logical name.

The default account associated with a system logical name applies unless an account is specified immediately after the logical name. For example, if the system logical name SCRACH were associated with account [100,100] on RP04 unit 2, opening the file SCRACH: [200,240] OTHER.DAT attempts to access the file OTHER.DAT on RP04 unit 2 under account [200,240]. The specifications SCRACH:OTHER.DAT and SCRACH:OTHER.DAT [200,240] refer to the file OTHER.DAT in account [100,100] on RP04 unit 2, the account associated with the logical name SCRACH.

The mount and dismount SYS calls create and delete entries in the first part of the logical name table. The mount call places a pack identification or logical name in the entry for the correct disk (unless NOLOGICAL was specified or unless the pack identification or logical name was already in use). The dismount call removes a pack identification or logical name from the entry for the correct disk.

A name or pack identification in the first part of the table can be changed or removed by the system logical name call. Entries in the second part of the table can be added or removed by the logical name SYS call. The following sections describe the variations of the logical name SYS call.

[1]The capability to define system logical names for nondisk devices and additional logical names for disks is optional on all RSTS/E systems. An attempt to use this capability on a system for which no system logical name table space has been configured always generates the NO ROOM FOR USER ON DEVICE error (ERR=4).

7.2.21.1 Add New Names – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(21%), the system logical name code
3	CHR$(1%) to add a new entry in the second part of the logical name table
4	Not used
5-6	Project-programmer number to be associated with this logical name. If these bytes are 0, no account is associated with the logical name.
7-10+	The system logical name, in Radix-50 format
11-22	Not used
23-26+	The device name and unit designation to which the logical name applies
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL FILE NAME No name is found in bytes 7 through 10 or the name found contains nonalphanumeric characters.	2
ACCOUNT OR DEVICE IN USE The name specified in bytes 7 through 10 duplicates one already in either the first or second part of the table.	3
NO ROOM FOR USER ON DEVICE All entries in the logical name table are in use. To free up an entry, issue the remove logical name SYS call.	4
NOT A VALID DEVICE The device specification given in bytes 23 through 26 is illegal or the related device is not configured on the system.	6

Discussion:

This call scans the entire system logical name table for the name given in bytes 7 through 10. The name cannot duplicate a logical name or pack identification defined in either the first or second part of the table. If the name does not exist in the table, the call adds an entry to the second part of the table.

7.2.21.2 Remove Logical Names – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(21%), the system logical name code
3	CHR$(0%) to remove a system logical name from either the first or second part of the logical name table
4-6	Not used
7-10+	The system logical name, in Radix-50 format.
11-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL FILE NAME No name is found in bytes 7 through 10 or the name found contains nonalphanumeric characters.	2
CAN'T FIND FILE OR ACCOUNT The name specified in bytes 7 through 10 is not currently defined as a logical name.	5

Discussion:

This call scans the entire system logical name table for the existence of the name specified in bytes 7 through 10. The call removes the logical name or pack identification from the first part of the table or removes an entire entry from the second part of the table.

7.2.21.3 Change Disk Logical Name – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(21%), the system logical name code
3	CHR$(255%) to change the logical name associated with a disk in the first part of the logical name table
4-6	Not used
7-10+	The system logical name, in Radix-50 format
11-22	Not used
23-26+	The name and unit designation of the disk device whose logical name is to be changed
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL FILE NAME No name is found in bytes 7 through 10 or the name found contains nonalphanumeric characters.	2
ACCOUNT OR DEVICE IN USE The logical name specified in bytes 7 through 10 duplicates one already in either the first or second part of the table.	3
CAN'T FIND FILE OR ACCOUNT The disk specified in bytes 23 through 26 is not configured on this system.	5
NOT A VALID DEVICE The device specified in bytes 23 through 26 is illegally formatted or is not a disk.	6

Discussion:

This call accesses the entry in the first part of the system logical name table for the disk specified in bytes 23 through 26. The logical name specified in bytes 7 through 10 is placed in the entry.

7.2.22 Add and Remove System Files – Privileged

Swapping files on RSTS/E are dynamically added at the start of time sharing and can be added and removed during time sharing. Swapping files must be removed to properly shut down time sharing. Optionally, two other system files, the overlay and error message files, can be added during time sharing to optimize system performance.

This SYS function adds and removes these system files. Through the INIT and UTILTY programs, a system manager can optionally create and add the swapping files and create other system files. The SHUTUP system program removes the files so that the disks on which they reside can be dismounted during the normal system shutdown. Refer to the *RSTS/E System Manager's Guide* for details on these operations. The following sections generally describe the operations and define the errors that can occur.

7.2.22.1 Add System Files – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(23%), the system file code
3	CHR$(N%) where N% designates the file to add as follows: 0 swapping file slot 0 1 swapping file slot 1 2 illegal - generates error 3 swapping file slot 3 4 overlay file 5 error message file
4	CHR$(1%), to add a system file
5-6	Not used
7-10+	To add a file which currently exists in account [0,1] and which has a .SYS extension, specify here the name, in Radix-50 format. If no name is given here (all bytes are zero), the add operation must be for a non-file structured disk to be used as a swapping device. If a file is specified, the system insures that it exists, is large enough and has proper characteristics.
11-22	Not used
23-26+	The name and unit designation of the device (must be disk) on which the file resides. If all bytes are zero, the public structure (SY:) is used.
27-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ILLEGAL FILE NAME No name is specified in bytes 7 through 10 when an overlay or an error message file is being added; or the name specified contains nonalphanumeric characters.	2
ACCOUNT OR DEVICE IN USE A swapping file is being added to a nonfile structured disk but the disk is currently mounted (that is, it is being used as a file structured device).	3
NO ROOM FOR USER ON DEVICE If an overlay or error file is being added, this error indicates that the file is not long enough. (The overlay file must be at least 32 blocks and	4

Meaning	ERR Value
the error file at least 8 blocks.) If a swapping file is being added to a file structured device, this error means that the file is not long enough to store even 1 job.	
CAN'T FIND FILE OR ACCOUNT A system file is being added to a file structured disk but the file with the name specified in bytes 7 through 10 and with a .SYS extension does not exist in account [0,1].	5
NOT A VALID DEVICE The device specified in bytes 23 through 26 is disk but is not configured on this system.	6
DEVICE NOT AVAILABLE A swapping file is being added to a nonfile structured disk but either the disk unit or its controller has been disabled. The system manager must use an initialization option to enable the unit or its controller.	8
PROTECTION VIOLATION A system file is being added to a file structured disk. Either the unit is logically write locked or the file specified in bytes 7 through 10 is bad (that is, it is not contiguous or is currently open).	10
NAME OR ACCOUNT NOW EXISTS The system file being added as described in byte 3 is already installed on the system.	16
ILLEGAL SYS( ) USAGE The number specified in byte 3 is either 2 or is greater than 5. The swapping file for slot 2 must exist on the system disk and cannot be added during time sharing. System files to be added are defined only by the values 0, 1, 3, 4, and 5.	18
DISK PACK IS NOT MOUNTED A system file is being added to a file structured disk but that disk is not currently mounted. Use either the INIT.BAC or UTILTY command MOUNT to logically mount the disk before the file is added.	21
DEVICE NOT FILE STRUCTURED The device specified in bytes 23 through 26 is not a disk device.	30

Discussion:

This SYS call to FIP either designates an entire disk to be added as a swapping slot or specifies a file to be added as a swapping slot, overlay, or error message file. By using the initialization options, the system manager creates system files in account [0,1] on a system disk or on nonsystem (public or private) disks. This call dynamically assigns system file space to provide flexibility in system operations.

The *RSTS/E System Generation Manual* discusses the rules and guidelines for planning and creating system files. Adding previously created system files with this call requires that the system manager plan his resources properly. For example, swapping files need contiguous space on a disk. If a file on disk is to be added for swapping, the system manager must have created thc contiguous space at the proper size. If a fixed head disk is to be added as a swapping slot, the system manager must ensure that the device is available for such usage.

Although this call adds swapping file space, it does not alter the job maximum allowed on the system. By adding a swapping file, a program merely increases the capability of the system to handle a larger number of jobs. To actually increase the job maximum after a swapping file is added, the enable logins SYS call must be issued or the LOGINS command of the UTILTY system program must be executed.

7.2.22.2 Remove System Files – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP
2	CHR$(23%), the system files code
3	CHR$(N%) where N% designates the file to remove as follows: 0 swapping file slot 0 1 swapping file slot 1 2 illegal - generates error 3 swapping file slot 3 4 overlay file 5 error message file
4	CHR$(0%) to remove a system file
5-30	Not used

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
ACCOUNT OR DEVICE IN USE The swapping file to be removed can be properly removed but currently contains one or more swapped out jobs. The system will lock the file and begin swapping jobs to other files. Retry the call at a later time when the swapped out jobs are no longer in this file.	3
PROTECTION VIOLATION A swapping file is to be removed but its removal will decrease the swapping file space below the limit required to store the maximum number of jobs on the system. To remove the swapping file, decrease the number of logins currently allowed (by either the SET LOGINS x command or SYS call). wait until the number of logged in jobs falls to the maximum, and try the removal operation again.	10
ILLEGAL SYS( ) USAGE The number specified in byte 3 is either 2 or is greater than 5. The swapping file for slot 2 must exist on the system disk and cannot be removed during time sharing. System files to be removed are defined only by the values 0, 1, 3, 4, and 5.	18

Discussion:

This SYS call to FIP removes a system file from operation. Removing previously added system files is required to shut down time sharing operations. Removing system files for other purposes allows a system manager to adjust system operation without ending time sharing. For example, if a fixed head disk, currently operating as a swapping device, malfunctions during time sharing, the system manager can decrease the allowed number of logins appropriately, remove the swapping slot for that device, dynamically add another device or file to replace the disk as a swapping slot and increase the new, allowed number of logins to take advantage of the added swapping space. Upon shutting down the system, the system manager can disable the malfunctioning unit by software to allow maintenance and to isolate the device from time sharing access. Normal time sharing operations can proceed without further alterations to the system.

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 memory and tables section of memory.[1] 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

[1] Accessing some device registers may cause unpredictable system results. PEEKing at device registers addressable via the UNIBUS is, therefore, not recommended.

7.3.1 Fixed Locations In Monitor

The information shown in Table 7-5 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.

Table 7-5 Monitor Fixed Locations

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)
522(word)	JOBF	Address of the JDFLG word in the job data block of the currently running job (always the user's own job data block)
524(word)	IOSTS	Address of the JDIOST (low) byte and JDPOST (high) byte in the job data block 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 second job data block (JDB2) of the job. The contents of the JDB2 bytes 24 and 25 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(PEEK(520%)+8%)+24%)
```

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%)))+6%) 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. To find the keyboard number, however, the program can use the FIP SYS call number 9 (return error message) described in Section 7.2.4.2.

CHAPTER 8
SYSTEM CALLS FOR LOCAL INTERJOB COMMUNICATION

8.1 LOCAL INTERJOB COMMUNICATION

Local communication between jobs running on a single RSTS/E system is a function of the Send/Receive facilities available in the RSTS/E Monitor.[1] Local senders can send messages (via the Send Local Data Message call) to local receivers. The receiver controls the communication by limiting the number of messages that can be queued and by declaring which senders are allowed to queue messages. The receiver passes this control information to the Monitor by means of a receiver declaration (via the Declare Receiver system call).[2] The system queues messages until the maximum number of messages specified by the receiver is pending for that particular receiver. Thereafter, any attempt by a local job to send another message to that receiver results in an error that is returned to the sender. In general, receivers must process pending messages frequently to avoid tying up system resources for lengthy periods of time. At the completion of message processing, a job must issue a Remove Receiver system call so that unwanted messages are not queued.

If DECNET/E is included in the system during system generation, a local job can use the network calls to communicate with other local jobs. In this case, the job functions as a network job. The use of network services to communicate with local jobs imposes DECNET/E restrictions and additional overhead. Use of the network calls, however, does allow programs to be coded and debugged locally before they are run on some other system in the network.

Every message, whether for local or for network communications, is divided into a parameter area and a data area. For a local message, the parameter area can contain from 0 to 20 bytes of user-defined data; the data portion can contain up to 512 bytes. Because the parameter area in a local message can contain user-defined data, the distinction between parameter and data is arbitrary for local messages. However, the distinction is important for a network message, in which the parameter area is used for DECNET information.

8.2 FORMAT OF THE SEND/RECEIVE SYSTEM CALLS

The general format of the system calls described in this chapter is:

```
V$=SYS(CHR$(6%)+CHR$(22%)+CHR$(S%)+ . . . +0$)
```

where:

V$	is the target string returned by the call.
=	is an assignment operator (the LET verb is implied).
SYS( )	indicates a system call.
CHR$(6%)	is the system function code for a call to the file processor (that is, the FIP call).
+	is the concatenation operator required between function, subfunction, and argument codes.

[1]Extensions to the system calls presented in this chapter are used in DECNET/E network communication. DECNET/E is the software package that extends RSTS/E to include network capabilities. The extended system calls used in network communication are described in the *RSTS/E DECNET User's Guide.*

[2]The receiving job must be privileged to issue a Declare Receiver system call.

CHR$(22%)	is the Send/Receive function code.
CHR$(S%)	is the user-specified subfunction code (for example, S%=1% indicates a Declare Receiver, S%=0% indicates a Remove Receiver system call).
. . .	indicates other arguments that must be specified for the system calls. The other arguments have the form CHR$(X%) where CHR$ is a function that converts data to character format, and X% is the user-specified argument defined by the specific system call.
O$	is optional user-defined data.

Throughout the system call descriptions, the following terms are used.

Term	**Meaning**
Reserved - must be zero.	This field is reserved for future use. The user must specify zero for each byte in the field. Trailing zeros need not be passed.
Ignored	This field is ignored by the system. The user can specify anything in the bytes in this field. However, it is a good programming practice to pass zeros in these bytes.
Not meaningful - should be ignored.	The bytes in this field do not contain useful information. In future releases, these bytes may have meaning. This term appears in the data returned by the various system calls.

NOTE

Unlike the SYS calls to FIP described in Chapter 7, the arguments passed to and returned from this Send/Receive call are longer than 30 bytes. Arrays used in CHANGE statements described in Sections 7.2.1 and 7.2.2 should be dimensioned to handle 40-byte strings.

Messages sent with the old format number 18 calls can always be received with this call. Messages sent with this call may be received with the number 18 call provided that the message is limited to the 20-byte parameter area passed in the SYS call string.

Because this call is a functional superset of the number 18 calls, this call should be used in all new applications.

8.3 DECLARE RECEIVER – PRIVILEGED

Data Passed:

Bytes	**Value**	**Meaning**
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(1%)	Declare Receiver subfunction code.

Bytes	**Value**	**Meaning**
4	CHR$(0%)	Reserved – must be zero.
5-10	Name	Receiver's logical name. The receiver's name is a 6-character, left-justified, ASCII string that is padded to six characters with spaces.
11-12	CHR$(0%)	Reserved – must be zero.
13-16		Ignored.
17-20	CHR$(0%)	Reserved – must be zero.
21	CHR$(0%)	Object Type Code. Object type codes are defined in the *RSTS/E DECNET User's Guide.* Legal values are 0 through 127. For local interjob communication, the object type code is zero unless DECNET/E is being used.
22	CHR$(L%+P%+N%+S%)	Access Control Field. This byte controls the types of senders that are allowed to queue messages for this job. It is the sum of the following four bit values:
	L%	Local/No Local Senders. If L%=0%, messages from local senders are not queued. Local senders who use network functions are considered network senders in this context. If L%=1%, messages from local senders are queued.
	P%	Local Privileged/Local Nonprivileged. This bit is ignored if L%=0% (no local senders allowed). If P%=0%, local senders do not have to be privileged to queue messages. If P%=2%, local senders must be privileged.
	N%	Network Logical Links/No Logical Links This bit controls the queuing of requests for DECNET/E logical links. The *RSTS/E DECNET User's Guide* describes network links. For local interjob communication, this bit should be zero.
	S%	Network Single Messages/No Single Messages. This bit controls the queuing of single network messages. For more information, see the *RSTS/E DECNET User's Guide*. For local interjob communication, this bit should be zero.

Bytes	**Value**	**Meaning**
23-24	B%	Buffer Maximum. The buffer maximum can be any positive value from one to 32767. A zero or negative value indicates that the monitor's buffer pool is not to be used for the data portion of messages. See the discussion below.
25	CHR$(M%)	Message maximum. The message maximum can be any value between one and 255. See the discussion below.
26	CHR$(L%)	Link maximum. The link maximum can have any value between 0 and 255. A job uses the link maximum to declare the maximum number of DECNET logical links it is willing to support at any one time. For local interjob communication, the link maximum should be zero.
27-28		Ignored.
29-40	CHR$(0%)	Reserved – must be zero.

Data Returned: No meaningful data are returned

Possible Errors:

ERR Value	**Meaning**
3	ACCOUNT OR DEVICE IN USE The calling job already exists in the receiver's list of declared receivers. This error may indicate a logic error in the program or that a previous program running under the same job number failed to remove itself from the receiver list before terminating. In the latter case, the calling job should remove itself (via a Remove call) and then reissue the declaration.
4	NO ROOM FOR USER ON DEVICE No small buffers are available to hold the arguments passed in the Declare Receiver call. Because the system's use of small buffers is dynamic, a subsequent retry may proceed without error.
10	PROTECTION VIOLATION The caller was nonprivileged at the time of the receiver declaration.
16	NAME OR ACCOUNT NOW EXISTS The logical name passed in Bytes 5-10 is being used by another job.

ERR Value	Meaning
18	ILLEGAL SYS( ) USAGE An inconsistency exists in the arguments passed.

Discussion:

A program identifies itself to the Monitor for message Send/Receive operations with a Declare Receiver call. The Monitor maintains a list of Receiver Id Blocks that hold the arguments passed in the receiver declaration, the message queue, and other system maintained information. A job can send local messages without performing a receiver declaration. However, to be eligible to receive messages, a job must have a Receiver Id Block.

The access control field (Byte 22) controls the types of network access permitted and the types of local senders permitted. The possible values are 0-15. However, for local interjob communication, only the values 1 (any local sender) and 3 (only privileged local senders) have meaning.

Each pending message in the system occupies system buffer space. One 16-word buffer from the Monitor's buffer pool is used for each message to hold the user- or DECNET-defined parameters and other system-specific information. Additional buffer space is needed for the data portion of the message.

The Monitor usually allocates buffer space from the extended buffer pool (if an extended buffer pool exists). If space in the extended buffer pool is not available, the Monitor's buffer pool is used. Because the Monitor's pool is a critical system resource, the job that must use the Monitor pool has to set a limit on the amount of space it will use. The buffer maximum (Byte 23-24) is a limit (in bytes) on the Monitor pool space that can be used for the data portion of messages.

The system maintains a count of messages queued for each receiver. The message maximum (Byte 25) limits the number of messages that will be queued for this receiver. This limit applies only to messages from local senders and network single messages. Local messages and network single messages are not queued unless the current count is less than the declared maximum. An error is returned to a local sender who attempts to send a message to a receiver whose count has exceeded this maximum.

8.4 SEND LOCAL DATA MESSAGE

Data Passed:

Bytes	Value	Meaning
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(-1%)	Send Local Data Message subfunction code.
4	CHR$(J%)	Job Number (times 2) of the local job to receive this message. If J% = 0, the call uses the logical name in Bytes 5 through 10 to determine the receiver. Otherwise, J% represents the job number (times 2) of the local receiving job. For example, if J%=8, the message is sent to job 4.

Bytes	Value	Meaning
5-10	N$	Receiver's logical name. The receiver's name is a 6-character, left-justified, ASCII string, that is padded to six characters with spaces. This field is used only if Byte 4 is CHR$(0%).
11	CHR$(C%)	Channel Number for the I/O buffer that contains the data portion of the message. If this byte is zero, a string beginning at Byte 41 contains the data portion of this message (if any). If this byte contains a channel number (any value from 1 to 12), a buffer defined by the length and offset values contains the data portion of this message. The message data (up to 512 bytes) should be left-justified in the buffer for channel C% beginning at the offset value defined in Bytes 15-16. Channel 0 can be used for the I/O buffer if 128 is added to the channel number; that is, CHR$(128%+0%). In general, CHR$(128%+C%) allows channels 0 through 12 to be used for I/O buffers.
12	CHR$(0%	Reserved – must be zero.
13-14	L%	Length (in bytes) of the message to send from the channel buffer. If Byte 11 is zero, the system ignores these bytes. For local data messages, this length field can have any value between zero and 512, subject to the restriction that the length of the message is less than or equal to the buffer size minus the offset value. If the length is zero, the system sends the whole buffer (i.e., from the offset to the end of the buffer).
15-16	O%	Offset value. If Byte 11 is zero, the system ignores these bytes. Offset from the beginning of the buffer where the message data begins. The offset must be in the range zero to <buffer size - 1>.
17-20	CHR$(0%)	Reserved - must be zero.
21-40	P$	Optional User Parameter String. A maximum of 20 bytes of user-defined data can be passed as parameters to the receiver of this message.
41+	D$	Optional Data String. A maximum of 512 bytes of user-defined data can be passed to the receiver's buffer. These bytes are ignored if Byte 11 is non-zero.

Data Returned: No meaningful data are returned.

Possible Errors:

ERR Value	**Meaning**
4	NO ROOM FOR USER ON DEVICE For local message operations, this error means that the number of messages pending for this receiver is at its declared maximum. The sender should try again later. If this error occurs frequently, the receiver is not processing its messages quickly enough.
5	CAN'T FIND FILE OR ACCOUNT For local messages, the receiving job, referenced by job number or logical name, was not found in the list of declared receivers. The receiving job must be declared (via the Declare Receiver system call) before any data can be transmitted.
9	I/O CHANNEL NOT OPEN The channel specified in Byte 11 of the data passed is not open. The job must open the channel and try again.
10	PROTECTION VIOLATION Some access violation has occurred. Either the sender is nonprivileged and the receiver requires senders to be privileged, or the receiver does not allow any local senders.
18	ILLEGAL SYS( ) USAGE The job number passed in Byte 4 is odd. Byte 4 must be zero or the receiver's job number times 2.
31	ILLEGAL BYTE COUNT FOR I/O The offset and/or length fields passed in Bytes 13-16 are illegal. The following relationships must be true for a send call. 1. The offset must be less than the buffer size. 2. The length must be less than or equal to the buffer size minus the offset value. The buffer size minus the offset value must be less than or equal to the maximum message length. The offset and length fields are checked for validity whenever a channel number is passed in Byte 11.
32	NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later retry may proceed without error.

Discussion:

A local job can send a message to a declared receiver by specifying either a job number or a logical name. When Byte 4 of the data passed is nonzero and even, it is interpreted as the job number (times 2) of the intended receiver. The logical name field is ignored. If Byte 4 is zero, the call attempts to send the message to the receiver whose logical name matches Bytes 5-10 of the data passed. Sending messages by job number is slightly more efficient than sending by logical name.

In a receiver declaration, the receiving job specifies the types of senders who are allowed to send messages. If no local senders are allowed, all attempts to send messages to the receiver will fail. Similarly, if local senders must be privileged, an attempt by a nonprivileged job to send a message to this receiver will also fail. All such access violations terminate with ERR=10 and the message is not sent.

8.5 RECEIVE

Data Passed:

Bytes	Value	Meaning
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(2%)	Receive subfunction code.
4	CHR$(S%+T%+L%+N%)	Modifier for this receive. The modifier is the sum of the following four values:
	S%	Sleep/No Sleep. If S%=0% and no messages are pending for this job, the receive call returns an immediate error (ERR=5). If S%=1%, the job sleeps until a message is queued. The duration of the sleep can be limited by Bytes 27-28. See the discussion for further details.
	T%	Truncate/No Truncate. If T%=0%, an attempt to receive a message that is too long for the buffer (indicated by Bytes 11-16 of the data passed) results in a partial message being transferred to the caller. The remainder of the message is saved and can be retrieved by subsequent Receive calls. If T%=2%, a message that is too long for the buffer (specified by Bytes 11-16 of the data passed) is truncated. For T%=0% or T%=2%, however, the number of bytes from the data portion of the message that was delivered to the buffer is returned in Bytes 13-14 of the data returned. The number of bytes remaining (T%=0%) or discarded (T%=2%) is noted in Bytes 9-10 of the data returned.

Bytes	Value	Meaning
	L%	Local Selection If L%=0%, local selection is disabled. Providing N% (described below) is also disabled, the first message on the receiver's queue of pending messages is delivered to the caller. If L%=4%, local selection is enabled. Only local messages are delivered on this Receive. The selection can be further qualified to a particular local sender by Bytes 5 and 6 described below.
	N%	Network Selection. If N%=0%, network selection is disabled. Providing L% (described above) is also disabled, the first message on the receiver's queue of pending messages is delivered to the caller. If N%=8%, network selection is enabled. Only network messages are delivered on this Receive. The selection can be further qualified to a particular DECNET logical link by specifying a DECNET user link address in Byte 5. **NOTE** If L%=4% and N%=8%, the local bit setting prevails; the network selective receive is ignored.
5	CHR$(S%)	Sender Selection. This byte is ignored if both L%=0% and N%=0%. Any nonzero value in this byte selects a particular sending job. Zero is a special case described for Byte 6. See the summary in the Discussion for meaningful combinations of Bytes 5 and 6. For local selection (L%=4% as described above), if this byte is equal to a job number times 2, the first message on the queue from that particular job is delivered to the caller. For network selection (N%=8% and L%=0%), if this byte is equal to a user link address, the first message on the queue from that particular DECNET logical link is delivered to the caller. Refer to the *RSTS/E DECNET User's Guide* for further details.
6	CHR$(255%)	Sender Selection Qualifier. This byte is ignored if both L%=0% and N%=0%. This byte is also ignored if byte 5 is non-zero. See the summary in the Discussion for meaningful combinations of bytes 5 and 6. For local selection, if byte 5 is zero and byte 6 is non-zero, this receive is requesting a message from the "system" (represented by job 0 which is not a real job number). This special case is intended

Bytes	Value	Meaning
		for use by ERRCPY which receives messages from the monitor error logging routines. The job number in these messages is zero. For network selection, if byte 5 is zero and byte 6 is non-zero, the receive is requesting a network single message. Network single messages are identified by a user link address of zero. Refer to the *RSTS/E DECNET User's Guide* for further details. If both byte 5 and byte 6 are zero, the selection bits (L% and N% as described above) select only the generic type of message to be delivered to the caller. For local selection (L%=4%), the first local message on the queue is delivered to the caller. For network selection (N%=8% and L%=0%), the first network message on the queue is delivered to the caller.
7-10		Ignored.
11	CHR$(C%)	Channel number for the I/O buffer to receive messages. If C% is between 1 and 12, the system return the data portion of the message in the buffer for channel C%. The channel must be open. If C%=0 or the buffer for channel C% is not large enough to accommodate the data portion of the message, the action taken depends on the value of the truncation bit in the receive modifier as described in the discussion below. Channel 0 can be used for the I/O buffer if 128 is added to the channel number; that is, CHR$(128%+0%). In general, CHR$(128%+C%) allows channels zero through 12 to be used for I/O buffers.
12	CHR$(0%)	Reserved - must be zero.
13-14	L%	Maximum message length (in bytes) desired on this receive. If Byte 11 is zero (i.e., no channel is specified), the offset and length fields are ignored. The length field limits the number of data bytes that are returned on this receive. If the length is zero, all buffer space remaining between the offset and the end of the buffer is considered available to receive a message. Otherwise, a maximum of L% bytes is returned on this receive. The specified length must be less than or equal to the buffer size minus the specified offset.
15-16	O%	Offset from the start of the buffer. The offset field determines where in the buffer the data portion of the message is returned. The offset value is added to the location of the beginning of the buffer. The offset value must be in the range 0 to <size of buffer - 1>.
17-20	CHR$(0%)	Reserved - must be zero.

Bytes	Value	Meaning
21-26		Ignored.
27-28	CHR$(T%)	Sleep time in seconds.
		If Byte 4 requests a sleep and no messages are pending, the sleep is terminated after T seconds. If T%=0%, the length of the sleep is indefinite; the job is not awakened until one of the three events (described in the discussion below) causes the job to be awakened. If the receive terminates because the sleep timer expires, an error (ERR=15) is returned.
29-40	CHR$(0%)	Reserved - must be zero.

Data Returned: Local Data Message

Bytes	Value	Meaning
1-2		Not meaningful - should be ignored.
3	CHR$(-1%)	Local Data Message subfunction code.
4	CHR$(J%)	Job number of the local sender.
		For local messages, this byte contains the job number (times 2) of the local sender.
5-6+	PPN	Project-programmer number.
7-8		Not meaningful - should be ignored.
9-10	R%	Number of bytes remaining in the data portion of the message.
	q	This is a count of bytes that were not delivered to the caller on this Receive. If truncation was not requested (T%=0% in Byte 4 of the data passed) and there are bytes in the buffer, the message remains queued. The rest of the data can be retrieved on subsequent Receive calls. If truncation was requested (T%=2% in Byte 4 of the data passed), the message is dequeued and this count is the number of bytes discarded.
11-12		Not meaningful - should be ignored.
13-14	L%	Length of the message transferred to the buffer.
		This count is the number of bytes actually transferred to the channel buffer on this Receive call. If no channel number was specified (Byte 11 = 0% in the data passed), this count is zero. In this case, the size of the data portion of the message is available in Bytes 9-10 of the data returned.
		Note that if the number of bytes transferred (Bytes 13-14 of the data returned) and the number of bytes remaining (Bytes 9-10

Bytes	Value	Meaning
		of the data returned) are both zero, the entire message consists of parameters that are available in Bytes 21-40 of the data returned.
15-20		Not meaningful – should be ignored.
21-40	P$	Use Parameter String. These bytes contain the data passed as parameters by the sender of this message. The system pads any unused bytes to a length of 20 bytes with zeros.

Possible Errors:

ERR Value	Meaning
5	CAN'T FIND FILE OR ACCOUNT If a Receive without sleep was issued, this error indicates that no messages are pending. If a Receive with sleep was issued, this error indicates that no messages were pending when the Receive call was issued. The error is returned when the job is awakened from the sleep. The program must execute the Receive again to retrieve any pending messages (see the discussion below).
9	I/O CHANNEL NOT OPEN An attempt was made to receive a message, but Channel C%, specified in Byte 11 of the data passed, is not open. The program must open the channel and try again.
15	KEYBOARD WAIT EXHAUSTED The sleep timer has expired. No messages were queued or delimiters typed for the duration of the sleep period.
18	ILLEGAL SYS( ) USAGE The job is not a declared receiver. Before any receive can succeed, the job must be entered in the receiver list.
31	ILLEGAL BYTE COUNT FOR I/O The offset and length fields passed in Bytes 13-16 are illegal. The following relationships must be true for a receive call. 1. The offset must be less than the buffer size. 2. The length must be less than or equal to the buffer size minus the offset value. The offset and length fields are checked for validity whenever a channel number is passed in Byte 11.

Discussion:

On any Receive call, the system checks the eligibility of the job to receive messages and returns ERR=18 if the job is not in the list of declared receivers. Passing this initial check, the call attempts to receive a message based on the receive modifier passed in Byte 4. Normally, a Receive call returns the first message on the receiver's queue of pending messages. The selective receive bits (L% and N% in Byte 4) can be used to select messages from particular senders identified by the local job number of DECNET user link address (as indicated by Byte 5). If the Sleep bit is off (S%=0% in Byte 4) and no messages are pending, the system generates ERR=5 and immediately passes the error to the calling program. If no messages are pending and the Sleep bit is on (S%=1% in Byte 4), the job is put into a sleep state (called a receiver sleep). The duration of the sleep can be limited by the sleep timer in Bytes 27-28 of the data passed.

A job in a receiver sleep can be awakened by any of three events:

1. A local or DECNET message is queued for the job. The job is awakened with an ERR=5.
2. The sleep timer expires. The job is awakened with ERR=15.
3. A delimiter is typed on any of the terminals owned by the job. The job is awakened with ERR=5.

In all three cases, the job is awakened with an error but is not passed a message. To obtain a pending message, the job must execute the Receive call again. Since the job may have been awakened by terminal input or expiration of the timer, a check for pending messages can be made by executing the Receive call without a sleep or by executing a terminal input operation using RECORD 8192% for immediate return.

The Receive call returns parameters in the target string and the data portion of the message (if any) in the I/O buffer specified by Byte 11. If the program must handle any DECNET/E messages, or local messages longer than the 20 bytes of user-defined parameters, a channel buffer must be available to receive the data portion of the message.

The number of bytes from the data portion of a message actually delivered to the buffer (if any), and the number of bytes remaining in the message (if any), can be determined from the data returned with the Receive call. Bytes 13-14 indicate the number of bytes from the data portion of the message that were delivered to the buffer.

The truncation bit (T%) in Byte 4 determines whether the remaining bytes in the channel buffer are retained or are discarded. If the user sets T%=0%, the remaining bytes are retained. If T%=2%, the remaining bytes are discarded. Bytes 9-10 indicate the number of bytes that remain to be transferred or were discarded (depending on the truncation bit in Byte 4).

When processing large message in small pieces, each successive Receive call retrieves a limited number of bytes from the same message. The normal sequence is to issue Receive calls until the number of bytes remaining in the data portion of the message is zero (as indicated by Bytes 9-10 of the data returned). The receiver then knows that the entire message has been delivered and dequeued.

A convenient method of assigning a buffer for message operations is to open the null device (NL:) at the desired buffer size using the RECORDSIZE clause on the OPEN statement. The null device is always available and can be opened as many times as required to obtain buffer space for any desired function. If a buffer is specified, the system ensures that the channel is open. If the channel is not open, the call results in an immediate ERR=9.

The program receiving a message selects the particular sender by combining receive modifier bits (L% and N% in byte 4) and the values of bytes 5 and 6. The possible combinations are summarized in Table 8-1.

Table 8-1 Sender Selection Summary

Data Passed				Result
Receive Modifier				
N%	L%	Byte 5	Byte 6	
0%	0%	-	-	Bytes 5 and 6 ignored; returns first queued message.
-	4%	0%	0%	Selects first local message.
		0%	nonzero	Selects job 0; used by error logging programs to select messages from monitor routines.
		nonzero	-	Selects local message by job number x2 in byte 5.
8%	0%	0%	0%	Selects first Network message.
		0%	nonzero	Selects first Network single message only; no link exists (uses Link Address=0).
		nonzero	-	Selects Network message by link (user Link Address) in byte 5.

8.6 REMOVE RECEIVER

Data Passed:

Bytes	Value	Meaning
1	CHR$(6%)	SYS call to FIP.
2	CHR$(22%)	Send/Receive function code.
3	CHR$(0%)	Remove subfunction code.
4	CHR$(J%)	Job Number (times 2) of the job to remove, or CHR$(0%) to remove the calling job. If J%=0%, the calling job need not be privileged. If J% is nonzero, the caller must be privileged.
5-40	CHR$(0%)	Reserved – must be zero.

Data Returned: No meaningful data are returned.

Possible Errors:

ERR Value	**Meaning**
10	PROTECTION VIOLATION The caller is nonprivileged and has attempted to remove another job (i.e., Byte 4 is non-zero).
18	ILLEGAL SYS( ) USAGE The job number passed in Byte 4 is odd. The job number must be zero to remove the caller, or job number times two to remove another job.

Discussion:

The Remove Receiver system call removes a job from the system's list of declared receivers. All pending messages are discarded. This call should be executed when message processing is being terminated but the job is to continue running. This prevents unwanted messages from accumulating in the queue of pending messages. Both the LOGOUT system program and the KILL command of UTILTY execute this call.

8.7 LOCAL SEND/RECEIVE EXAMPLES

Several examples of the Send/Receive SYS calls are presented in this section. The examples include a Receive Declaration, two Send Local Data calls, five Receives to demonstrate some of the possible options, and a Remove Receiver call. The series of examples is a program which can be run to demonstrate the operation of the Send/Receive functions. The examples are coded for illustration rather than efficiency. They do not handle all possible error conditions, and do not present all possible options. The examples should, however, give the general flavor of the services offered.

8.7.1 Declare Receiver Example

The following Receiver Declaration will establish the caller as a message receiver with the logical name "DEMO". Only local privileged senders will be allowed to send messages to this receiver. If Monitor buffer pool space is required to store pending messages destined for this receiver, a maximum of 1024 bytes of Monitor pool space (equivalent to 32 small buffers at 32 bytes each) will be used on this receiver's behalf before senders receive an error (ERR=4). Providing the buffer maximum is not exceeded (or extended pool space is available to store pending messages), up to 5 messages will be queued for this receiver before senders receive an error (also ERR=4). Finally, no request for DECNET logical links will be honored for this receiver.

```
10      EXTEND
900     DIM X%(40%)
1000    !
        ! RECEIVER DECLARATION EXAMPLE
        !
1110    LOGNAME$ = 'DEMO  '      !THIS RECEIVER'S LOGICAL NAME
1120    OBJTYPE% = 0%            !ALL ACCESS BY LOGICAL NAME
1130    ACCESS%  = 1%+2%         !ONLY LOCAL PRIVILEGED SENDERS ALLOWED
1140    BMAX%    = 1024%         !UP TO 1024 BYTES FROM MONITOR POOL
1150    MMAX%    = 5%            !OR UP TO 5 MESSAGES
1160    LMAX%    = 0%            !NO DECNET LOGICAL LINKS
1190    !
1200    X$   = SYS(CHR$(6%)+CHR$(22%)+CHR$(1%)+CHR$(0%)
              + LOGNAME$      + STRING$(10%,0%)
              + CHR$(OBJTYPE%)
              + CHR$(ACCESS%)
```

```
                  + CHR$(BMAX%)    + CHR$(SWAP%(BMAX%))
                  + CHR$(MMAX%)
                  + CHR$(LMAX%))
```

8.7.2 Send Local Data Examples

The following local send calls will send a message from a string and from a buffer. In both cases, the receiver is referenced by its logical name. The intended receiver is the receiver whose logical name is "DEMO". Note that if this series of examples were run as a program, the job would be sending messages to itself.

The first example is a send from a string. There is no need to distinguish between "parameter" and "data" areas of the message as long as the receiver is aware that part of the message will be delivered in the target string returned by the SYS call and the remainder will be returned in a specified buffer.

```
2000    !
        ! LOCAL SEND EXAMPLES
        !
2100    ! THE FIRST SEND IS A SIMPLE STRING SEND
        !
2110    MSG1$ = "THIS MESSAGE WAS SENT FROM A STRING."
2190    !
2200    X$   = SYS(CHR$(6%)+CHR$(22%)+CHR$(-1%)+CHR$(0%)
                 + LOGNAME$        + STRING$(10%,0%)
                 + MSG1$)
        !
2210    PRINT "1ST MESSAGE SENT = ";MSG1$
```

The second send call will send a message from a buffer. In this case, the null device is opened on channel 2 to obtain buffer space, the message data is loaded into the buffer using LSET, and the data portion of the message is sent from the buffer. User defined "parameters" are also included with this message and will be delivered to the receiver. The use of JUNK$ at the beginning of the buffer illustrates the use of the buffer offset field in Send calls.

```
2300    !
        ! THE SECOND SEND IS A SEND FROM A BUFFER
        !
2310    CHANNEL% = 2%
2320    OPEN "NL:" AS FILE CHANNEL%, RECORDSIZE 100%
2330    FIELD #CHANNEL%, 10% AS JUNK$, 90% AS TEXT$
2340    MSG2$    = "THIS MESSAGE WAS SENT FROM A BUFFER."
2350    PARAM$   = "MESSAGE # 2 ..."
2360    MSGLEN% = LEN(MSG2$)
2370    OFFSET% = LEN(JUNK$)
2380    LSET TEXT$ = MSG2$
2390    !
2400    X$   = SYS(CHR$(6%)+CHR$(22%)+CHR$(-1%)+CHR$(0%)
                 + LOGNAME$
                 + CHR$(CHANNEL%)+ CHR$(0%)
                 + CHR$(MSGLEN%) + CHR$(SWAP%(MSGLEN%))
                 + CHR$(OFFSET%) + CHR$(SWAP%(OFFSET%))   + STRING$(4%,0%)
                 + PARAM$)
        !
2410    PRINT "2ND MESSAGE SENT = ";MSG2$
2420    PRINT "PARAMETERS  SENT = ";PARAM$
2430    PRINT
```

8.7.3 Receive Examples

Five Receive examples are presented below. If this series of examples were run as a program, the series of receives would retrieve the two messages sent in the Send examples of Section 8.7.2.

The first receive is a simple receive into a buffer large enough to hold any expected message. The receiver is willing to wait up to 10 seconds for a message, so the sleep bit in the receive modifier is turned on, and a 10 second limit is passed as the sleep timer. Truncation is also requested since no messages are expected which will be larger than the buffer available. In this example, the ON ERROR GOTO which would normally be used to field the "sleep expired" error (ERR=15), is omitted for simplicity. As mentioned in the discussion of the first send example, part of the message is delivered to the receiver as "parameters" in the target string, and the remainder of the message is delivered to the channel buffer.

```
3000    !
        ! RECEIVE EXAMPLES
        !
3100    ! THIS FIRST RECEIVE WILL RECEIVE THE FIRST MESSAGE SENT
        !
3110    FIELD #CHANNEL%, 100% AS TEXT$
3120    S% = 1% \ TIMER% = 10%          !REQUEST MAX 10 SECOND SLEEP
3130    T% = 2%                          !REQUEST TRUNCATION
3190    !
3200    X$  = SYS(CHR$(6%)+CHR$(22%)+CHR$(2%)
                + CHR$(S%+T%)    + STRING$(6%,0%)
                + CHR$(CHANNEL%)+ STRING$(15%,0%)
                + CHR$(TIMER%)   + CHR$(SWAP%(TIMER%)))
        !
3210    CHANGE X$ TO X%                  !MAKE TARGET STRING USEABLE
3220    MSGLEN% = X%(13%)+SWAP%(X%(14%))!LENGTH OF RECEIVED MESSAGE
3230    BYTREM% = X%(9%) +SWAP%(X%(10%))!BYTES LOST DUE TO TRUNCATION
3240    IF BYTREM% <> 0% THEN STOP       !CANNOT OCCUR IN EXAMPLE
3250    FIELD #2%, MSGLEN% AS MSG$       !FIELD FOR LENGTH RECEIVED
3260    PRINT "MESSAGE RECEIVED = ";RIGHT(X$,20%);MSG$ !PRINT RCVD  MSG
```

The next receive is used to determine the sender's job number and length of the next pending message. An indefinite length sleep is requested to wait for a message to be queued. In this case no buffer is provided since we choose not to receive the data portion of any message on this call.

```
3300    !
        ! THIS SECOND RECEIVE CALL IS USED TO DETERMINE IF ANY FURTHER
        ! MESSAGES ARE PENDING AND TO DETERMINE THE JOB NUMBER OF THE
        ! SENDER FOR SUBSEQUENT SELECTIVE RECEIVE EXAMPLES.
        !
3310    S% = 1% \ TIMER% = 0%           !REQUEST INDEFINITE SLEEP
3320    T% = 0%                          !NO TRUNCATION ALLOWED NOW
3390    !
3400    X$ = SYS(CHR$(6%)+CHR$(22%)+CHR$(2%)+CHR$(S%))
        !
3410    CHANGE X$ TO X%
3420    SNDJOB% = X%(4%)                 !GET SENDING JOB NUMBER * 2
3430    BYTREM% = X%(9%)+SWAP%(X%(10%)) !GET # BYTES IN DATA PORTION
3440    IF BYTREM% = 0% THEN STOP        !IMPOSSIBLE IN THIS EXAMPLE
```

The third receive illustrates sender selection. For purposes of the example, assume the second message sent above is the only message pending (which is the case for this series of examples). If the receive selects some other sender (SNDJOB%+2% in the example below) and no sleep is requested, an error (ERR=5) should result as demonstrated below. Note that truncation is not allowed on this receive since we want to preserve the pending message.

```
3500    !
        ! THE THIRD RECEIVE SELECTS MESSAGES FROM A PARTICULAR
        ! SENDER. IN THIS EXAMPLE A RANDOM JOB IS SELECTED TO
        ! FORCE AN ERROR.
        !
3510    ON ERROR GOTO 3620
3520    LCLSEL% = 4%                       !REQUEST LOCAL SELECTION
3590    !
3600    X$   = SYS(CHR$(6%)+CHR$(22%)+CHR$(2%)
                + CHR$(LCLSEL%)
                + CHR$(SNDJOB%+2%))
        !
3610    STOP                               !CANNOT OCCUR IN THIS EXAMPLE
3620    IF ERR <> 5% THEN STOP             !ERR 5 WAS INTENTIONAL
3630    RESUME 3700
```

We know the sender's job number and the number of bytes in the next pending message. If buffer space is restricted for some reason, it may be necessary to retrieve the message in several pieces. For the example, the receive below arbitrarily restricts the number of bytes the caller will accept to 20 bytes by using the length field in the receive call.

```
3700    !
        ! THE NEXT RECEIVE SELECTS THE SENDER DETERMINED ABOVE.
        ! ONLY A PORTION OF THE MESSAGE IS RETRIEVED ON THIS CALL.
        !
3710    MAXLEN% = 20%                      !LENGTH ARBITRARILY RESTRICTED
3790    !
3800    X$   = SYS(CHR$(6%)+CHR$(22%)+CHR$(2%)
                + CHR$(LCLSEL%)
                + CHR$(SNDJOB%) + STRING$(5%,0%)
                + CHR$(CHANNEL%)+ CHR$(0%)
                + CHR$(MAXLEN%) + CHR$(SWAP%(MAXLEN%)))
        !
3810    CHANGE X$ TO X%                    !MAKE TARGET STRING USEABLE
3820    IF X%(4%) <> SNDJOB% THEN STOP     !CANNOT OCCUR IN THIS EXAMPLE
3830    MSGLEN% = X%(13%)+SWAP%(X%(14%))!GET LENGTH RECEIVED
3840    BYTREM% = X%(9%) +SWAP%(X%(10%))!GET COUNT NOT DELIVERED
3850    IF BYTREM% = 0% THEN STOP          !CANNOT OCCUR IN THIS EXAMPLE
```

At this point the program has received part of the message (MSGLEN% characters), and the remainder of the message (BYTREM% characters) are still queued. The last receive will retrieve the remainder of the message and place it in the buffer immediately after the portion delivered on the previous receive. Sender selection ensures that the data received is the remainder of the same message delivered on the previous call.

```
3900    !
        ! THE LAST RECEIVE WILL RETRIEVE THE REST OF THE DATA FROM
        ! THE SECOND MESSAGE SENT IN LINE 2400 ABOVE.
        !
3910    OFFSET% = MSGLEN%                  !BUFFER OFFSET FOR RECEIVE
3990    !
4000    X$   = SYS(CHR$(6%)+CHR$(22%)+CHR$(2%)
                + CHR$(LCLSEL%)
                + CHR$(SNDJOB%) + STRING$(5%,0%)
                + CHR$(CHANNEL%)+ STRING$(3%,0%)
                + CHR$(OFFSET%) + CHR$(SWAP%(OFFSET%)))
        !
4010    CHANGE X$ TO X%                    !MAKE TARGET STRING USEABLE
4020    IF X%(4%) <> SNDJOB% THEN STOP     !CANNOT OCCUR IN THIS EXAMPLE
4030    MSGLEN%=MSGLEN%+X%(13%)+SWAP%(X%(14%)) !TOTAL LENGTH OF MSG
4040    BYTREM%=X%(9%)+SWAP%(X%(10%))      !AND COUNT NOT DELIVERED
```

```
4050    IF BYTREM% <> 0% THEN STOP          !WHICH SHOULD BE ZERO
4060    FIELD #2%,MSGLEN% AS MSG$           !FIELD COMPLETE MESSAGE
4070    PRINT "MESSAGE RECEIVED = ";MSG$!AND PRINT COMPLETE MESSAGE
```

On the last three receive calls, the examples have been working on a single pending message. Recall that the data portion of the message was sent from a buffer and was just received in a buffer. However, the second send in Section 8.7.2 also included some "parameters" which were actually delivered to the receiver on each of the last three receives. This can be verified at this point by merely printing the last 20 characters of the target string returned by the last receive call.

```
4100    !
        ! PRINT PARAMETER AREA OF SECOND MESSAGE FOR VERIFICATION
        !
4110    PRINT "PARAMETER AREA    = ";RIGHT(X$,20%)
```

The example concludes with a Remove Receiver call and a close of the channel buffer used to receive messages.

```
5000    !
        ! REMOVE RECEIVER EXAMPLE
        !
5200    X$ = SYS(CHR$(6%)+CHR$(22%)+CHR$(0%)+CHR$(0%))
6000    !
6010    CLOSE 2%
32767   END
```

8.7.4 Summary of Data Values

Figure 8-1 is provided as a summary of the data passed and returned in the Send/Receive calls.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41+
SEND LOCAL DATA (from buffer)	6	22	–1	0	RECEIVER'S LOGICAL NAME						C	0	LENGTH		OFFSET		0	0	0	0	USER-DEFINED PARAMETERS (P$)																				
SEND LOCAL DATA (from string)	6	22	–1	0	RECEIVER'S LOGICAL NAME						0	0	IGNORED				0	0	0	0	USER-DEFINED PARAMETERS (P$)																				[D$]
SEND LOCAL DATA (from buffer)	6	22	–1	J	IGNORED						C	0	LENGTH		OFFSET		0	0	0	0	USER-DEFINED PARAMETERS (P$)																				
SEND LOCAL DATA (from string)	6	22	–1	J	IGNORED						0	0	IGNORED				0	0	0	0	USER-DEFINED PARAMETERS (P$)																				[D$]
REMOVE RECEIVER	6	22	0	0,J	0	0	0	0	0	0	0	0	IGNORED				0	0	0	0	RESERVED – MUST BE ZERO																				
DECLARE RECEIVER	6	22	1	0	RECEIVER'S LOGICAL NAME						0	0	IGNORED				0	0	0	0	OBJ. TYPE	ACCESS	BMAX		MMAX	LMAX	IGNORED		RESERVED – MUST BE ZERO												
RECEIVE	6	22	2	MOD	SNDR SEL	SEL QUAL	IGNORED				C	0	LENGTH		OFFSET		0	0	0	0	IGNORED						SLEEP TIMER		RESERVED – MUST BE ZERO												
RECEIVE	6	22	2	MOD	SNDR SEL	SEL QUAL	IGNORED				0	0	IGNORED				0	0	0	0	IGNORED						SLEEP TIMER		RESERVED – MUST BE ZERO												
RESERVED	6	22	3																																						
LOCAL DATA RETURNED ON RECEIVE	IGNORED		–1	J	PPN		IGNORED		BYTES REMAINING		IGNORED		LENGTH		IGNORED						P$ = USER-DEFINED PARAMETERS																				▨

Figure 8-1 Summary of Send/Receive Data

CHAPTER 9
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.

1. 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.
2. 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.

9.1 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.

A DATA statement within these lines contains a full command definition, the file specification of the program to run, a number denoting the minimum number of characters necessary to abbreviate the command, and a condition code. If the leftmost characters typed match the command definition or an allowable abbreviation, LOGIN removes those characters from the command line. LOGIN puts the remainder of the command line in core common and chains to the program specified in the DATA statement. The line number to which LOGIN chains depends on the code in the DATA statement. If the code is 4, LOGIN chains to line 32000 of the program. Line 32000 is the standard entry point for running programs from a logged out terminal.

Thus, for SY 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.

```
32260 DATA SYSTAT, $SYSTAT, 2,4
```

The 2 in the DATA statement allows SY to be recognized as an abbreviation for SYSTAT. Allowable abbreviations are SY, SYS, SYST, and SYSTA. LOGIN removes SY from the command line and writes the remainder of the command line in core common. Because of the 4 in the DATA statement, LOGIN chains to line 32000 of SYSTAT.BAC stored in account [1,2]. When SYSTAT runs, it reads core common to obtain the command line to execute.

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 with the program name because there is no default account when running at a terminal not logged into the system.

```
    32190 DATA WHAT, "[2,2]HELP", 2,4
```

The 2 in the DATA statement allows a 2-character minimum abbreviation WH.

After LOGIN is recompiled and stored in the system library account with protection code <232>, the program HELP under account [2,2] starts running at line 32000 whenever, WH, WHA, or WHAT is typed at a logged out terminal. If the program is to be run only from logged out terminals, it need not be compiled with protection code <232>.

NOTE

This feature is not the same as the concise command language facility. LOGIN is run by the RSTS/E monitor. The ability to chain to other programs is a feature of LOGIN.

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 full privileges. To implement protection, the program itself must perform the protection check. When a logged out job terminates, it can simply transfer control to the END statement. The run-time system automatically prints the BYE message and removes the program from memory.

9.2 DESIGNING A PROGRAM TO RUN BY A CCL COMMAND

Using the CCL command in the UTILTY system program, the system manager can define special commands. These special commands are called concise command language commands. When typed at a logged in terminal, a CCL command loads and runs a program from a predefined account and device. This mechanism enables users on a system to run specially tailored programs by commands similar to monitor commands.

To validate a CCL command, a run-time system passes a typed string to the RSTS/E monitor. The CCL command parser, located in the monitor, processes the string. If a string passed to the CCL parser is not a valid CCL command, control is returned to the run-time system. If the string is a valid CCL command, the monitor sets up the job core common area, extracts information from predefined CCL data, sets up the program to run, and transfers control to the run-time system associated with that program. The run-time system then runs the program.

This discussion of CCL describes two separate operations: how the BASIC-PLUS run-time system interprets keyboard commands and how the RSTS/E CCL command parser interacts with BASIC-PLUS to execute CCL commands. With this information, an applications programmer can design programs to run by CCL commands.

The predefined CCL data resides in a linked list of monitor buffers. Because CCL commands do not have permanent definitions, it is recommended in the *RSTS/E System Manager's Guide* that all CCL commands be defined at system start up time. At the start of time sharing, INIT.BAC sets up the CCL data from commands given in the start up control file. Even the CCL commands for programs designed by DIGITAL must be defined for each time sharing session. These CCL commands are listed in Section 3.1 of the *RSTS/E System Manager's Guide.*

9.2.1 BASIC-PLUS and CCL Commands

The precedence of CCL commands is above that of BASIC-PLUS commands and immediate mode statements and is below that of line numbered statements. Consequently, BASIC-PLUS scans input at a terminal at command level and applies the following rules.

1. If the input begins with numbers, the line is passed to the BASIC-PLUS syntax analyzer for processing and storage as intermediate code.
2. If the line begins with nonnumeric characters, the line is passed to the CCL command parser for processing and validation.
3. If the line does not contain a valid CCL command, the CCL parser passes it to the BASIC-PLUS syntax analyzer for immediate mode execution.
4. If the line does not contain a valid command or immediate mode statement, BASIC-PLUS generates an error message.

Because line-numbered input is processed first, its interpretation is not altered by CCL command definitions. A CCL command duplicating either a valid BASIC-PLUS command or immediate mode statement, however, overrides that command or statement. The system runs the appropriate program and thereby destroys the current contents of the job area.

An example can demonstrate the difference between a CCL and BASIC-PLUS commands. The command CAT and the CCL command DIR both print directory listings. CAT is recognized by BASIC-PLUS as a command. The run-time system calls monitor code which produces the listing. The user's job space is not disturbed. The DIR command, however, loads and runs the BASIC-PLUS program DIRECT from the system library to create a listing. The user's job space is overwritten by the DIRECT program.

9.2.2 CCL Syntax and Switches

The following lines show the proper syntax for a valid CCL command called COMMAND which can be abbreviated COM.

```
COM[M[A[N[D]]]] [<switch(es)>]  [/<anything>]
COM[M[A[N[D]]]] [<switch(es)>]  [<space><anything>]
```

(The definition of the CCL command is made by the UTILTY system program at the start of time sharing.)

The CCL command parser passes one of the following forms of the parsed command in the job core common area.

```
COMMAND [/<anything>]
COMMAND [<space><anything>]
```

Note that the CCL parser always expands each command to its fully defined form.

The optional value <switch(es)> is stripped from the command line by the parser. The command line can contain one or both of two switches in the format shown below.

```
[<space>]/SI[Z[E]]:[+]  [#]<digit-list>[.]
```

where:

/SI	denotes the size in K words the program must expand to.
:	terminates the /SIZE switch.
+	designates an increment in size over the program's usual size. Without the plus character, the digit-list is the total size in K words the program should expand to.

#	indicates the digit-list value is given in octal. The default radix is decimal.
<digit-list>	is the value for size, in K words. Size can be neither less than 1 nor greater than 28 (decimal).
	explicitly indicates decimal radix for digit-list.

[<space>] /DET [A [C [H]]]

where:

/DET	indicates that the program is to be run detached from the job console terminal.

The monitor extracts these switches and sets status bits for the run-time system. It takes no action on the switches. The run-time system optionally interprets and processes the status information.

These CCL switches can occur in any order but must immediately follow the CCL command. If a syntax error is detected, the parser generates either the ILLEGAL SWITCH USAGE or the ILLEGAL NUMBER error. If the command line exceeds 127 characters (the maximum capacity of core common), the parser generates the LINE TOO LONG error.

Because the parser searches the typed line for special switches, it is recommended that programmers not define program switches which conflict with either /SI or /DET. If such switches are defined, special instructions are required for their use. For example, to have a /SI or /DET switch passed to a program, it must be preceded in the command line by text not beginning with a / character (such as SY: or another device specification).

9.2.3 CCL Command Line Parsing

The CCL parser performs the following operations on a command string passed to it.

1. translates the string
2. checks the string for a valid CCL command
3. writes the fully expanded CCL command into core common
4. checks the remaining string for both of the valid CCL switches
5. writes the remaining line (excluding CCL switches) to core common
6. sets up the CCL program to run
7. sets a flag from data in the CCL command definition block
8. passes control back to the program's run-time system which, in turn, runs the program (at the appropriate line number for BASIC-PLUS programs)

Actions taken by the run-time system are independent of what the CCL parser does. (The BASIC-PLUS actions are described in Section 9.2.4.)

To translate the command line, the parser performs several steps as follows.

1. For all characters in the input string, it trims the parity bit, discards end-of-line and excess (NUL and RUBOUT) characters, and discards leading and trailing space and tab characters.
2. For the remaining characters not within quotation marks, it changes all tab characters to spaces, reduces adjacent spaces to a single space, discards all control characters, and converts lower case alphabetic characters (CHR$(97) to CHR$(122)) to upper case equivalents (CHR$(65) to CHR$(90)).
3. Characters within quotation marks are not altered.

The parser scans the leftmost part of the translated command line for a potential CCL command. The scan of the translated string terminates on the first occurrence of one of the following characters.

1. end-of-line (RETURN, LINE FEED, or ESC)
2. slant (/)
3. space

The scanned string is compared with each entry in the list of valid CCL commands. If the scanned string matches a defined CCL command at its abbreviation point or matches any part of a defined CCL command beyond the abbreviation point, the parser writes the fully expanded CCL command to the job core common area. If no match is found, the parser writes the translated command line to core common and returns control to the run-time system.

Because of the translation, spaces typed within a CCL command itself are critical. A space typed within the CCL command itself terminates the scan of the command line. For example, if the CCL command COMMAND (with abbreviation COM) is typed COM<SPACE>MAND, the parser interprets the COM as a legitimate abbreviation for COMMAND and handles <space>MAND as part of the line to pass to the program. Likewise, the typed command line CO<space>MMAND would not be matched with a command whose minimum abbreviation is COM.

When the parser ascertains that the translated string contains a valid CCL command, it starts moving the CCL command string to the job's core common area. Any errors encountered later by the parser result in random contents in the core common area.

If the remaining part of the translated string begins with either a slant character or a space followed by a slant character, the parser checks for a valid CCL switch. If a valid CCL switch is found, the parser checks further for another adjacent switch. Duplication of a switch generates the ILLEGAL SWITCH USAGE error. Any CCL switches found are extracted from the command line. The parser writes the remaining part of the command line to core common. If any error is found in the CCL command or switches, the parser stops processing the command line and returns control to the run-time system with an error indication.

After processing the command line, the monitor sets up the related CCL program to run. A flag from the CCL command definition is passed to the run-time system. The fully defined CCL command and the remaining string are passed in the core common area.

9.2.4 BASIC-PLUS Action

BASIC-PLUS receives control from the CCL parser at one of two points. If the command is not a valid CCL command or if it generated an error, control is returned in line. When the monitor fails to validate a CCL command, BASIC-PLUS processes the translated string for execution in immediate mode. Should the parser return an error, BASIC-PLUS prints the related message and the READY prompt. When the monitor does validate a CCL command, it passes control to the run-time system to run a BASIC-PLUS program. BASIC-PLUS takes the following actions.

1. sets the STATUS variable
2. checks the line number at which the program is to be entered
3. checks whether the program is to be detached

Based on the results of the above actions, BASIC-PLUS runs the program.

BASIC-PLUS sets the STATUS variable according to the rules shown in Table 9-1.

If BASIC-PLUS finds that the line number is nonzero, it checks the flag passed by the monitor. If the job is not permanently privileged, BASIC-PLUS permanently drops temporary privileges unless the CCL definition indicated that privileges are to be kept for this program. This action prevents a job from bypassing a program's protection mechanism by entering a program at a line other than the lowest numbered one. If the job has permanent privileges and the /DET switch was specified, BASIC-PLUS detaches the job and closes all channels on which the console terminal is open. No error is generated if the job does not have a permanent privilege and the /DET switch was specified.

Table 9-1 STATUS Variable After CCL Entry

Bit	Test	Meaning
0-7	(STATUS AND 255%)	If bit 13 is 0, this byte must be 0. If bit 13 is 1, this byte is the size value n (in decimal) passed in the /SIZE:n switch or is -n to indicate that the size value was passed in the /SIZE:+n switch (the plus character preceded the size value). (To determine whether the size value is negative, check the most significant bit by the (STATUS AND 128%) test.)
8-12		Reserved for future use.
13	(STATUS AND 8192%)	If the /SIZE:n switch was specified, this bit is 1. Otherwise, it is 0.
14	(STATUS AND 16384%)	If the /DET switch was specified, this bit is 1. Otherwise, it is 0.
15	(STATUS < 0%)	This bit is always 1 (the value of STATUS is always negative).

BASIC-PLUS takes no action for the /SIZE:n switch. For run-time systems other than BASIC-PLUS, this switch is a signal that an increase in job size is required. Such run-time systems do not perform dynamic memory expansion, as BASIC-PLUS does, and do require the switch to expand job size.

9.2.5 Conventions Used by BASIC-PLUS Programs

As a convention, BASIC-PLUS programs supplied by DIGITAL and invoked by standard CCL commands reserve lines 30000 to 30999 for CCL routines. These routines extract the parsed command line passed in core common, check for errors, and dispatch to other routines within the program. This convention allows programs to discover that they were entered by means of CCL. The programs execute the SYS call to get the core common string and scan the string for the specific CCL command expanded by the CCL parser. This action also allows a single program to be run by one of several CCL commands. Upon determining which CCL command caused them to run, the programs dispatch to routines to process the remainder of the command line.

In designing a program to run by CCL command, it must be remembered that a CCL command which is a subset of a BASIC-PLUS command or immediate mode statement supersedes that command or statement. For example, if PRINT or any subset of PRINT (P, PR, PRI, AND PRIN) is defined as a CCL command, the immediate mode PRINT statement is not executed.

9.3 CHANGING LOGIN TO SET A DIFFERENT SWAP MAXIMUM

The LOGIN system program sets the swap maximum to 28K words for all users. This action means that all users run with a swap maximum of 28K words.

To modify the LOGIN.BAS program to set a swap maximum less than 28K words for nonprivileged accounts, alter the J% = 28% 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% = 28%
        \ J% = 28% IF (A% AND -256%) = 256%
        \ I$ = SYS(CHR$(6%)+CHR$(-13%)+CHR$(-1%)+
               CHR$(-1%)+CHR$(-2%)+
               CHR$(-1%)+CHR$( 6%)+
               CHR$(-1%)+CHR$( J%))
        \ RETURN
```

Change the value 28% in the statement J% = 28% to any value less than or equal to the current default swap maximum used at system start up time.

To set a swap maximum less than 28K words for privileged accounts, change the J% = 28% expression in the second physical line at line number 15010. The statement on this line checks for a one as the project number of the account.

Recompile and reprotect the program on the system library account with protection code <232> as follows.

```
COMPILE SYO: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.

9.4 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.

9.4.1 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 in a single line number saves statement header space. Placing one statement per physical line improves readability.

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 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 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 more quickly because 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 2-byte integer variables and constants when possible; denote them with the percent sign (%). The subscripts in arrays should always be 2-byte integers, specified with % signs. Use 2-byte 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 UNTIL should be used when possible to replace loops. The statement,

```
10 IF X% < L% THEN X% = X% * X%
     \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, uses 35% less memory 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
```

9.4.2 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 (by the OPEN FOR INPUT statement) without wasting disk space, fragmenting the directory structure 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.

9.4.3 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 (do not 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"
```

A$ ——► "ABCD"

C$ ——► "EFG"

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$
```

A$ ——► "ABCD"

C$ ——► "ABC"

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$ + ""
```

A$ ——► "ABCD"

C$ ——► "ABCD"

LSET and RSET statements move data; LET statements move pointers. Proper use of LSET and RSET prevents generation of wasted space, and executes 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$(X$)

        \X$   = LEFT(X$,LEN(X$)-1%)

              WHILE RIGHT(X$,LEN(X$)) <= " "

                  AND LEN(X$)>0%

1010      FNT$ = X$

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.

9.5 CONVERTING NUMERIC DATA

A BASIC-PLUS program stores numeric data in memory in either floating point or 2-byte integer format. When performing input and output of the numbers with Record I/O, a program manipulates the data in an intermediate I/O buffer. Because BASIC-PLUS addresses data in the buffer through character definitions, the program must convert numeric data to character format for transfer to the I/O buffer before the output operation. Likewise, numeric data input by the program for processing in memory must be converted to numeric format after extraction from the I/O buffer.

The CVT functions are provided to perform the required format conversions. These functions are implemented for speed rather than for logical ordering. They use stack operations to convert data and, hence, reverse the expected ordering of bytes. In converting 2-byte integer data, the reversal merely transposes the high and low bytes of the word. In converting floating point data, however, the high and low bytes of each word are transposed and the ordering of the words is exactly reversed.

The reversal is usually not evident to the program. A program manipulating data with Record I/O executes one function before loading the data into the I/O buffer and later executes a related function upon reading the data back.

Figure 9-1 shows the conversion of 2-byte integer data by the CVT functions.

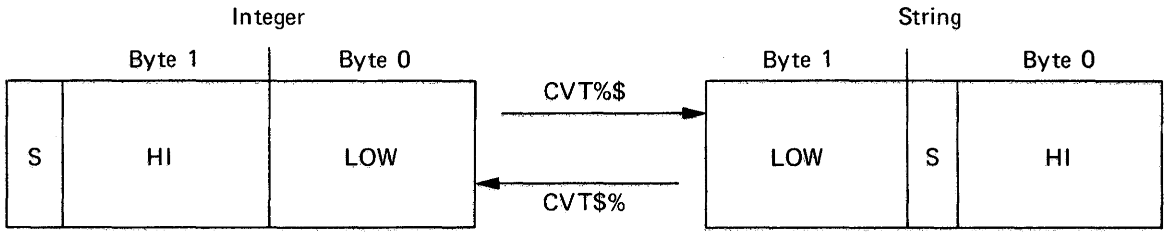


Figure 9-1 CVT Conversion of 2-byte Integer Data

The CVT%$( ) function reverses the byte order of the integer data word. The related function CVT$%( ) reverses the byte order of the character data, thus returning the integer to its correct byte order.

This reversal is evident, however, when a program uses Record I/O to read data not written by Record I/O. Under such circumstances, the data read into the buffer is in correct byte order. The CVT$% function reverses the correct byte order of the data in the buffer. A program must execute the SWAP%( ) function to put the bytes in the proper order. For example, the system writes the standard PDP-11 internal representation of the date as a 2-byte integer value on a DOS magtape label. A program, reading the label using non-file structured Record I/O, accesses the date in two bytes of a buffer addressed by D$. The CVT$%( ) function, as shown below, converts the string D$ to the 2-byte integer format.

```
10 D% = SWAP%(CVT$%(D$))
20 PRINT DATE$(D%)
```

The CVT$%( ) function reverses the correct order of the bytes. The SWAP%( ) function swaps the high and low bytes to place them in proper order. The integer D% is afterwards used in the DATE$( ) function to produce the standard printed format for the date.

Figure 9-2 shows the conversion of 2-word floating point data by the CVT functions.

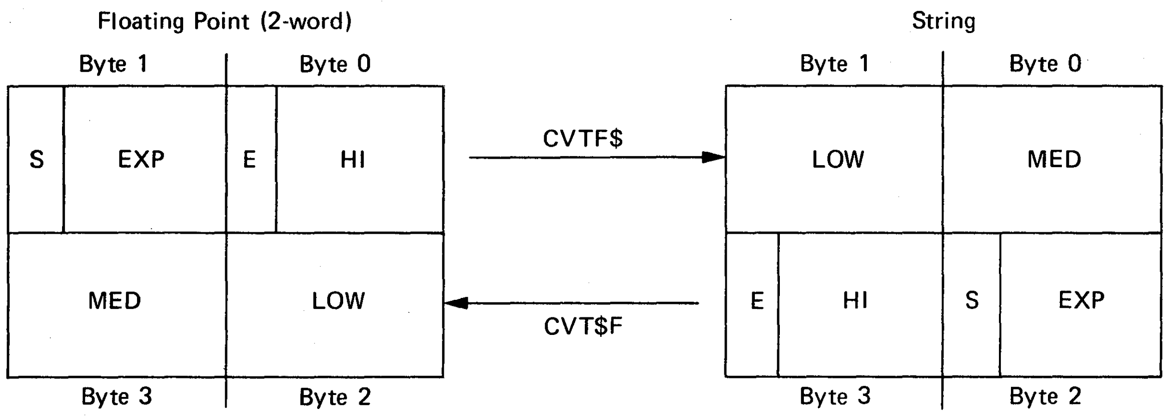


Figure 9-2 CVT Conversion of 2-word Floating Point Data

When converting a 2-word floating point value to a 4-byte character string, the CVTF$( ) function stacks the bytes in the reverse of their original order. The sign and exponent bits are not in the standard format. The same reversal occurs when the CVTF$( ) function converts a 4-word floating point value to an 8-byte character string. The results of the 4-word case are shown in Figure 9-3.

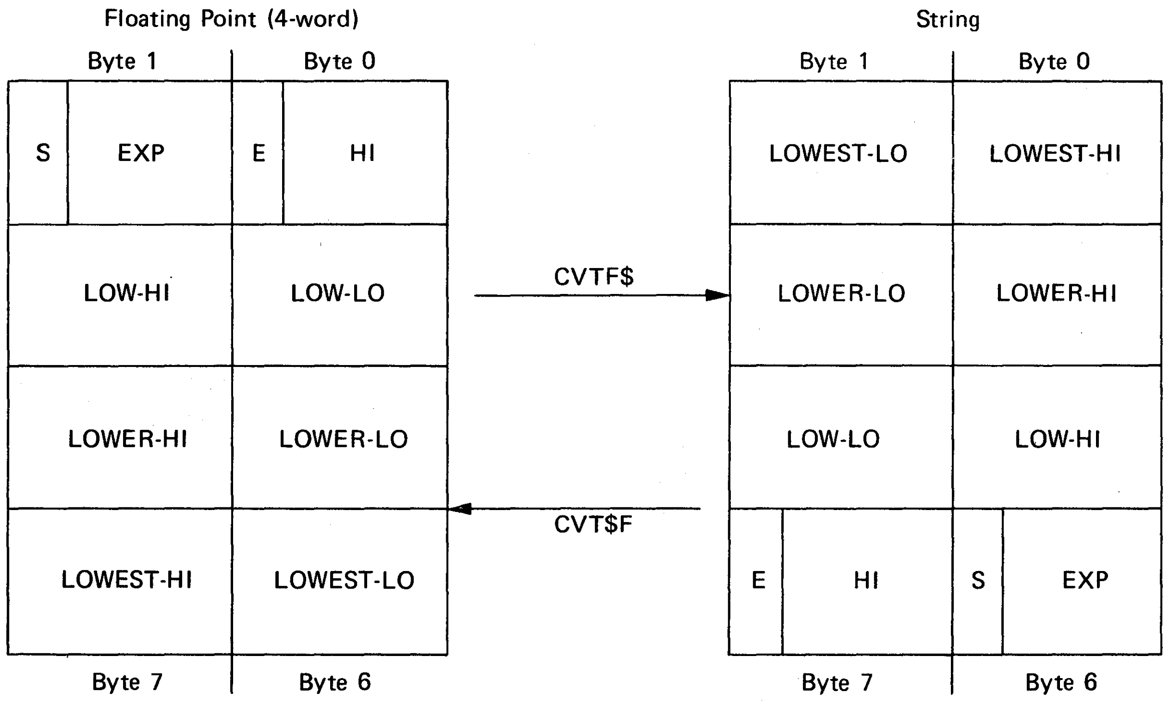


Figure 9-3 CVT Conversion of 4-word Floating Point Data

Normally the conversion makes no difference because a program converts numeric data using one function and converts it back again using the related function.

One occasion for confusion occurs, however, if a program reads floating point or 2-byte integer data from a virtual core array file using Record I/O operations. Because the virtual array processor neither requires conversion of data nor itself converts data during input and output operations, the virtual array data read into an I/O buffer by Record I/O statements is in correct byte order. In usual Record I/O operations, a program simply converts the character data in the buffer using the proper CVT function. But when a program performs this usual conversion on virtual array data, the CVT function reverses the correct byte order and the resulting numeric data is bad.

A solution to reading virtual array numeric data by Record I/O operations is to reverse the byte order of the data in the buffer before converting it.

NOTE

Reading a virtual array file using Record I/O is not a recommended programming practice because the usefulness of the automatic virtual array addressing mechanism is lost.

The sample program in Figure 9-4 creates a virtual array file and reads back the numeric data, reverses the byte order of the data in the buffer, and generates the numeric representation using the CVT function.

```
100     OPEN 'VIRT.DAT' FOR OUTPUT AS FILE 1%
        \ DIM #1, A(0) \ A,A(0) = RND * 1000.0
        \PRINT A
        \ CLOSE 1%
        \OPEN 'VIRT.DAT' FOR INPUT AS FILE 2%
220     L% = LEN(CVTF$(1.0))
        \ GET #2%
240     B$ = ''
        \ FOR I% = L% - 1% TO 0% STEP -1%
260             FIELD #2%, I% AS B1$, 1% AS B1$
280             B$ = B$ + B1$
        \ NEXT I%
300     PRINT CVT$F(B$);'SHOULD EQUAL';A
32767   CLOSE 1%,2%
        \ END
```

Figure 9-4 Sample Conversion of Virtual Array Data

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.

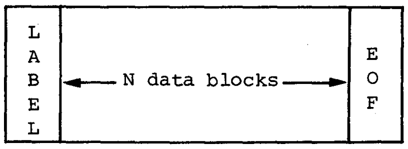


Figure A-1 A DOS Magtape File

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).

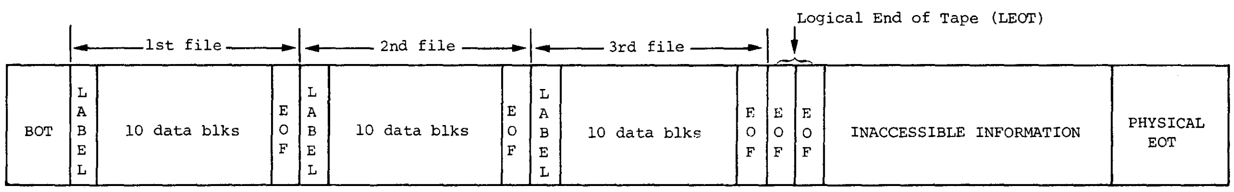


Figure A-2 DOS Magtape Consisting of 3 Files of 10 Data Blocks Apiece

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/E and DOS-11 use different protection code values. RSTS/E 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.

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.

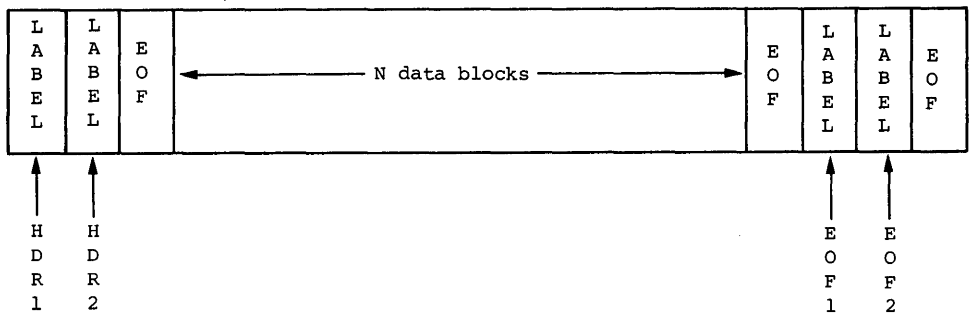


Figure A-3 An ANSI Magtape File

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).

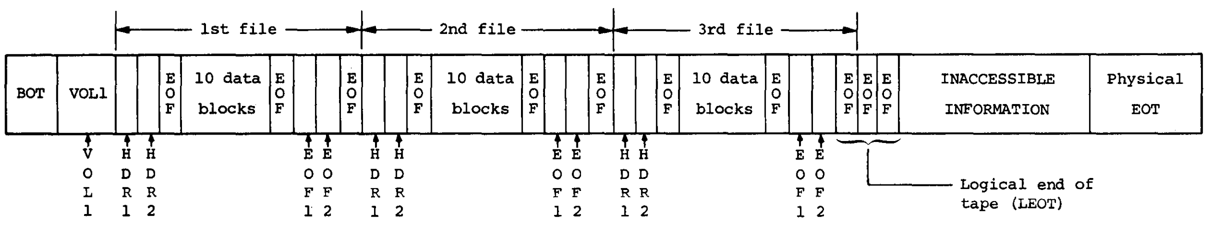


Figure A-4 ANSI Magtape Consisting of 3 Files of 10 Data Blocks Apiece

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. The characters D%B4431 define the corporation (D%=DIGITAL), the computer (B=PDP11), and a protection code which RSTS/E does not use.

A.2.1.2 Header 1 Label (HDR1) – Table A-3 shows the character position, field name and contents of each type 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

(continued on next page)

Table A-3 Header 1 Label Format (Cont.)

Character Position	Field Name and RSTS/E Usage	Contents
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

(continued on next page)

Table A-4 Header 2 Label Format (Cont.)

Character Position	Field Name and RSTS/E Usage	Contents
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 character precedes and CR character follows 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

(continued on next page)

Table A-5 End of File (EOF) 1 Record Format (Cont.)

Character Position	Field Name and RSTS/E Usage	Contents
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

(continued on next page)

Table A-6 End of File (EOF) 2 Record Format (Cont.)

Character Position	Field Name and RSTS/E Usage	Contents
		= (Space) means LF character precedes and CR character follows 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 17.1.5 of the *RSTS/E 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/E 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/E 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. Table B-1 shows the card codes.[1]

Table B-1 Card Reader Codes

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

(continued on next page)

[1]The DEC029 code used by RSTS/E complies with the ANSI standard for punched card codes. IBM uses a card code which differs in three characters. If the DEC029 card code is configured on the system, a patch can be installed to change the decode table so that cards punched for use on IBM equipment can be read without misinterpretation. Refer to the *RSTS/E Release Notes* for information on the codes affected.

Table B-1 (Cont.) Card Reader Codes

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
ERROR MESSAGES

Messages in RSTS/E are generated for BASIC-PLUS errors[1] and RSTS/E errors. To avoid confusion, both types of messages are called RSTS/E error messages and are described as one set. The BASIC-PLUS errors cover compiler and run time conditions such as a violation of the syntax rules (SYNTAX ERROR) and referencing an element of an array beyond the defined limits (SUBSCRIPT OUT OF RANGE). The RSTS/E errors involve operating system conditions such as failing to locate the file or account specified (CAN'T FIND FILE OR ACCOUNT) and requesting the hardware to perform a function for which it is not ready (DEVICE HUNG OR WRITE LOCKED).

In most cases, if no error trapping is being done (that is, an ON ERROR GOTO statement is not in effect), BASIC-PLUS stops running the program. It prints the error message and the line number of the BASIC-PLUS statement that was being executed when the error occurred. The following sample printout shows the procedure.

```
10        OPEN 'Z' FOR INPUT AS FILE 1%
RUNNH
?CAN'T FIND FILE OR ACCOUNT AT LINE 10

READY
```

As the READY message indicates, control returns to the system.

An exception to this procedure occurs when an INPUT statement is being executed at the job's console terminal and error trapping is not in effect. The system generates the error message and executes the statement again as shown in the sample printout below.

```
10        ON ERROR GOTO 0 \ INPUT 'INTEGER VALUE';A%
RUNNH
INTEGER VALUE? C
%DATA FORMAT ERROR AT LINE 10
INTEGER VALUE?
```

With error trapping disabled at line 10, an invalid response to the INPUT statement causes the system to print the error message, clear the error condition, and execute the statement again.

Associated with each message is an error variable called ERR. Whenever an error occurs with trapping in effect, the system checks the error variable which is a decimal number in the range 0 to 127. An error with a number between 1 and 70 causes the system to transfer control to the line number indicated in the ON ERROR GOTO statement. The system does not print the error message. The user program is able to check the ERR variable and perform a recovery procedure. If the error number is between 71 and 127, the system does not transfer control to the recovery routine but prints the message and returns control to the system. (Error number 0 is reserved to identify the system installation name.)

Because a BASIC-PLUS program can recover from certain errors, this appendix lists errors in two categories – recoverable and non-recoverable. The recoverable error messages are listed in ascending order of their related error numbers. A program can use these error numbers to differentiate errors. Non-recoverable errors are in alphabetical order without error numbers because a program can not use these numbers in an error handling routine.

[1]Different messages are generated while a job is operating under run-time systems other than BASIC-PLUS. Such run-time systems are those for COBOL and FORTRAN-IV. For these error messages, consult the appropriate User's Guides.

The first character position of each.message indicates the severity of the error. Table C-1 describes this standard.

Table C-1 Severity Standard in Error Messages

Character	Severity	Meaning
%	Warning	Execution of the program can continue but may not generate the expected results.
?	Fatal	Execution cannot continue unless the user removes the cause of the error.
	Information	A message beginning with neither a question mark nor a percent is for information only.

The severity indication is useful for utility programs such as BATCH which examines system output.

In the descriptions of error messages, certain abbreviations, as shown in Table C-2, denote special characteristics of the error.

Table C-2 Special Abbreviations for Error Descriptions

Abbreviation	Meaning
(C)	Continue. If an ON ERROR GOTO statement is not in effect, execution continues but with the conditions described.
(SPR)	Software Performance Report. This error should occur only under the conditions described. If it occurs under any other conditions, the user should file an SPR with DIGITAL and document the conditions under which the error occurred.

An error whose description is accompanied by the abbreviation (C) indicates an exception to the error trapping procedure. If such an error occurs in a program with no error trapping in effect, BASIC-PLUS prints the error message and line number but continues running the program. The following sample printout shows the procedure.

```
100     ON ERROR GOTO 0 \ A% = 32768.
200     PRINT A%
RUNNH
%INTEGER ERROR AT LINE 100
 0

READY
```

The INTEGER ERROR is generated at line 100 by the attempt to compute a value outside the range for integers. After the error message is printed, processing continues but with the conditions described in the error meaning. 0 is substituted for the erroneously computed value.

The number of RSTS/E error messages is restricted to 127. Because of this restriction, certain error messages have multiple meanings. The specific meaning of an error message depends on the operation being performed when the error condition occurs. For example, if the system attempts a file access and the designated file can not be located, RSTS/E generates the CAN'T FIND FILE OR ACCOUNT error (ERR=5). That same error condition, however, applies to other, generically similar access operations. Thus, if a program attempts to send a message to another program and the proper entry is not found in the system table of eligible receivers, RSTS/E returns error number 5. Though the second failure does not involve a file access error, it too is classified as an access failure.

Certain RSTS/E errors, although classified as user recoverable, are not capable of being trapped by a program. Table C-3 lists such errors.

Table C-3 Non-Trappable Errors in Recoverable Class

ERR	Message Printed
34	RESERVED INSTRUCTION TRAP
36	SP (R6) STACK OVERFLOW
37	DISK ERROR DURING SWAP
38	MEMORY PARITY FAILURE

These errors involve special conditions which a user program cannot control and which ought not to occur on a normal system. For example, the DISK ERROR DURING SWAP error indicates a hardware problem. The system does not return control to the program. The error condition itself, however, can be either transient or recurring. Such errors should be brought to the attention of the system manager for further investigation. These errors are recoverable in the strict sense that the monitor can take corrective action but the BASIC-PLUS run-time system does not return control to the user program.

C.1 USER RECOVERABLE

ERR	Message Printed	Meaning
0	(system installation name)	The error code 0 is associated with the system installation name and is used by system programs to print identification lines.
1	?Bad directory for device	The directory of the device referenced is in an unreadable format. The magtape label format on tape differs from the system-wide default format, the current job default format, or the format specified in the OPEN statement. Use the ASSIGN command to set the correct format default or change the format specification in the MODE option of the OPEN statement.
2	?Illegal file name	The filename specified is not acceptable. It contains unacceptable characters or the filename specification format has been violated. The CCL command to be added begins with a number or contains a character other than A through Z, 0 through 9 and commercial at (@).
3	?Account or device in use	Reassigning or dismounting of the device cannot be done because the device is open or has one or more open files. The account to be deleted has one or more files and must be zeroed before being deleted. The run time system to be deleted is currently loaded in memory and in use. Output to a pseudo keyboard cannot be done unless the device is in KB wait state. An echo control field cannot be declared while another field is currently active. The CCL command to be added already exists.

ERR	Message Printed	Meaning
4	?No room for user on device	Storage space allowed for the current user on the device specified has been used or the device as a whole is too full to accept further data.
5	?Can't find file or account	The file or account number specified was not found on the device specified. The CCL command to be deleted does not exist.
6	?Not a valid device	The device specification supplied is not valid for one of the following reasons. The unit number or its type is not configured on the system. The specification is logical and untranslatable because a physical device is not associated with it.
7	?I/O channel already open	An attempt was made to open one of the twelve I/O channels which had already been opened by the program. (SPR)
8	?Device not available	The specified device exists on the system but a user's attempt to ASSIGN or OPEN it is prohibited for one of the following reasons. The device is currently reserved by another job. The device requires privileges for ownership and the user does not have privilege. The device or its controller has been disabled by the system manager. The device is a keyboard line for pseudo keyboard use only.
9	?I/O channel not open	Attempt to perform I/O on one of the twelve channels which has not been previously opened in the program.
10	?Protection violation	The user was prohibited from performing the requested operation because the kind of operation was illegal (such as input from a line printer) or because the user did not have the privileges necessary (such as deleting a protected file).
11	?End of file on device	Attempt to perform input beyond the end of a data file; or a BASIC source file is called into memory and is found to contain no END statement.
12	?Fatal system I/O failure	An I/O error has occurred on the system level. The user has no guarantee that the last operation has been performed. This error is caused by hardware condition. Report such occurrences to the system manager. (See the discussion at beginning of appendix.)
13	?User data error on device	One or more characters may have been transmitted incorrectly due to a parity error, bad punch combination on a card, or similar error.
14	?Device hung or write locked	User should check hardware condition of device requested. Possible causes of this error include a line printer out of paper or high-speed reader being off-line.
15	?Keyboard WAIT exhausted	Time requested by WAIT statement has been exhausted with no input received from the specified keyboard.
16	?Name or account now exists	An attempt was made to rename a file with the name of a file which already exists, or an attempt was made by the system manager to insert an account number which is already within the system.

ERR	Message Printed	Meaning
17	?Too many open files on unit	Only one open DECtape output file is permitted per DECtape drive. Only one open file per magtape drive is permitted.
18	?Illegal SYS( ) usage	Illegal use of the SYS system function.
19	?Disk block is interlocked	The requested disk block segment is already in use (locked) by some other user.
20	?Pack IDs don't match	The identification code for the specified disk pack does not match the identification code already on the pack.
21	?Disk pack is not mounted	No disk pack is mounted on the specified disk drive.
22	?Disk pack is locked out	The disk pack specified is mounted but temporarily disabled.
23	?Illegal cluster size	The specified cluster size is unacceptable. The cluster size must be a power of 2. For a file cluster, the size must be equal to or greater than the pack cluster size and must not be greater than 256. For a pack cluster, the size must be equal to or greater than the device cluster size and must not be greater than 16. The device cluster size is fixed by type.
24	?Disk pack is private	The current user does not have access to the specified private disk pack.
25	?Disk pack needs 'CLEANing'	Non-fatal disk mounting error; use the CLEAN operation in UTILTY.
26	?Fatal disk pack mount error	Fatal disk mounting error. Disk cannot be successfully mounted.
27	?I/O to detached keyboard	I/O was attempted to a hung up dataset or to the previous, but now detached, console keyboard for the job.
28	?Programmable ↑C trap	A CTRL/C combination was typed while an ON ERROR GOTO statement was in effect and programmable CTRL/C trapping was enabled.
29	?Corrupted file structure	Fatal error in CLEAN operation.
30	?Device not file structured	An attempt is made to access a device, other than a disk, DECtape, or magtape device, as a file-structured device. This error occurs, for example, when the user attempts to gain a directory listing of a non-directory device.
31	?Illegal byte count for I/O	The buffer size specified in the RECORDSIZE option of the OPEN statement or in the COUNT option of the PUT statement is not a multiple of the block size of the device being used for I/O, or is illegal for the device. An attempt is made to run a compiled file which has improper size due to incorrect transfer procedure.
32	?No buffer space available	The user accesses a file and the monitor requires one small buffer to complete the request but one is not currently available. If the program is sending messages, two conditions are possible. The first occurs when a program sends a message and the receiving program has exceeded the pending message limit. The second occurs when a sending program attempts to send a message and a small buffer is not available for the operation.

ERR	Message Printed	Meaning
33	?UNIBUS timeout fatal trap	This hardware error occurs when an attempt is made to address non-existent memory or an odd address using the PEEK function. An occurrence of this error message in any other case is cause for an SPR.
34	?Reserved instruction trap	An attempt is made to execute an illegal or reserved instruction or an FPP instruction when floating point hardware is not available. (See discussion at beginning of appendix.)
35	?Memory management violation	This hardware error occurs when an illegal Monitor address is specified using the PEEK function. Generation of the error message in situations other than using PEEK is cause for an SPR.
36	?SP (R6) stack overflow	An attempt to extend the hardware stack beyond its legal size is encountered. (See discussion at beginning of appendix.) (SPR)
37	?Disk error during swap	A hardware error occurs when a user's job is swapped into or out of memory. The contents of the user's job area are lost but the job remains logged into the system and is reinitialized to run the NONAME program. Report such occurrences to the system manager. (See discussion at beginning of appendix.)
38	?Memory parity failure	A parity error was detected in the memory occupied by this job. (See discussion at beginning of appendix.)
39	?Magtape select error	When access to a magtape drive was attempted, the selected unit was found to be off line.
40	?Magtape record length error	When performing input from magtape, the record on magtape was found to be longer than the buffer designated to handle the record.
41	?Non-res run-time system	The run time system referenced has not been loaded into memory and is therefore non-resident.
42	?Virtual buffer too large	Virtual core buffers must be 512 bytes long.
43	?Virtual array not on disk	A non-disk device is open on the channel upon which the virtual array is referenced.
44	?Matrix or array too big	In-core array size is too large.
45	?Virtual array not yet open	An attempt was made to use a virtual array before opening the corresponding disk file.
46	?Illegal I/O channel	Attempt was made to open a file on an I/O channel outside the range of the integer numbers 1 to 12.
47	?Line too long	Attempt to input a line longer than 255 characters (which includes any line terminator). Buffer overflows.
48	%Floating point error	Attempt to use a computed floating point number outside the range 1E-38 $< n <$ 1E38 excluding zero. If no transfer to an error handling routine is made, zero is returned as the floating point value. (C)

ERR	Message Printed	Meaning
49	%Argument too large in EXP	Acceptable arguments are within the approximate range -89<arg<+88. The value returned is zero. (C)
50	%Data format error	A READ or INPUT statement detected data in an illegal format. For example, 1..2 is an improperly formed number, and 1.3 is an improperly formed integer, and X" is an illegal string. (C)
51	%Integer error	Attempt to use a computed integer outside the range -32768<n<32767. For example, an attempt is made to assign to an integer variable a floating point number outside the integer range. If no transfer to an error handling routine is made, zero is returned as the integer value. (C)
52	?Illegal number	Integer overflow or underflow or floating point overflow. The range for integers is -32768 to +32767; for floating point numbers, the upper limit is 1E38. (For floating point underflow, the FLOATING POINT ERROR (ERR=48) is generated.)
53	%Illegal argument in LOG	Negative or zero argument to LOG function. Value returned is the argument as passed to the function. (C)
54	%Imaginary square roots	Attempt to take square root of a number less than zero, The value returned is the square root of the absolute value of the argument. (C)
55	?Subscript out of range	Attempt to reference an array element beyond the number of elements created for the array when it was dimensioned.
56	?Can't invert matrix	Attempt to invert a singular or nearly singular matrix.
57	?Out of data	The DATA list was exhausted and a READ requested additional data.
58	?ON statement out of range	The index value in an ON-GOTO or ON-GOSUB statement is less than one or greater than the number of line numbers in the list.
59	?Not enough data in record	An INPUT statement did not find enough data in one line to satisfy all the specified variables.
60	?Integer overflow, FOR loop	The integer index in a FOR loop attempted to go beyond 32766 or below -32767.
61	%Division by 0	Attempt by the user program to divide some quantity by zero. If no transfer is made to an error handler routine, a 0 is returned as the result. (C)
62	?No run-time system	The run-time system referenced has not been added to the system list of run time systems.
63	?FIELD overflows buffer	Attempt to use FIELD to allocate more space than exists in the specified buffer.
64	?Not a random access device	Attempt to perform random access I/O to a non-random access device.

ERR	Message Printed	Meaning
65	?Illegal MAGTAPE ( ) usage	Improper use of the MAGTAPE function.
66	?Missing special feature	User program employs a BASIC-PLUS feature not present on the given installation.
67	?Illegal switch usage	A CCL command contains an error in an otherwise valid CCL switch. (For example, the /SI:n switch was used without a value for n or a colon; or more than one of the same type of CCL switch was specified.) A file specification switch is not the last element in a file specification or is missing a colon or an argument.

C.2 NON-RECOVERABLE

Message Printed	Meaning
?Arguments don't match	Arguments in a function call do not match, in number or in type, the arguments defined for the function.
?Bad line number pair	Line numbers specified in a LIST or DELETE command were formatted incorrectly.
?Bad number in PRINT-USING	Format specified in the PRINT-USING string cannot be used to print one or more values.
?Can't compile statement	
?Can't CONTinue	Program was stopped or ended at a spot from which execution cannot be resumed.
?Data type error	Incorrect usage of floating-point, integer, or character string format variable or constant where some other data type was necessary.
?DEF without FNEND	A second DEF statement was encountered in the processing of a user function without an FNEND statement terminating the first user function definition.
?End of statement not seen	Statement contains too many elements to be processed correctly.
?Execute only file	Attempt was made to add, delete or list a statement in a compiled (.BAC) format file.
?Expression too complicated	This error usually occurs when parentheses have been nested too deeply. The depth allowable is dependent on the individual expression.
?File exists-RENAME/REPLACE	A file of the name specified in a SAVE command already exists. In order to save the current program under the name specified, use REPLACE, or use RENAME followed by SAVE.
?FNEND without DEF	An FNEND statement was encountered in the user program without a previous function call having been executed.

Message Printed	Meaning
?FNEND without function call	A FNEND statement was encountered in the user program without a previous DEF statement being seen.
?FOR without NEXT	A FOR statement was encountered in the user program without a corresponding NEXT statement to terminate the loop.
?Illegal conditional clause	Incorrectly formatted conditional expression.
?Illegal DEF nesting	The range of one function definition crosses the range of another function definition.
?Illegal dummy variable	One of the variables in the dummy variable list of user-defined function is not a legal variable name.
?Illegal expression	Double operators, missing operators, mismatched parentheses, or some similar error has been found in an expression.
?Illegal FIELD variable	The FIELD variable specified is unacceptable.
?Illegal FN redefinition	Attempt was made to redefine a user function.
?Illegal function name	Attempt was made to define a function with a function name not subscribing to the established format.
?Illegal IF statement	Incorrectly formatted IF statement.
?Illegal in immediate mode	User issued a statement for execution in immediate mode which can only be performed as part of a program.
?Illegal line number(s)	Line number reference outside the range $1<n<32767$.
?Illegal mode mixing	String and numeric operations cannot be mixed.
?Illegal statement	Attempt was made to execute a statement that did not compile without errors.
?Illegal symbol	An unrecognizable character was encountered. For example, a line consisting of a # character.
?Illegal verb	The BASIC verb portion of the statement cannot be recognized.
%Inconsistent function usage	A function is defined with a certain number of arguments but is elsewhere referenced with a different number of arguments. Fix the reference to match the definition and reload the program to reset the function definition.
%Inconsistent subscript use	A subscripted variable is being used with a different number of dimensions from the number with which it was originally defined.
x(y)K of memory used	Message printed by the LENGTH command. The value for x is the current size, to the nearest 1K-word increment, of the program in memory. The value for y is the size to which the program can expand, given the run time system being used and the job's private memory size maximum set by the system manager.

Message Printed	Meaning
?Literal string needed	A variable name was used where a numeric or character string was necessary.
?Matrix dimension error	Attempt was made to dimension a matrix to more than two dimensions, or an error was made in the syntax of a DIM statement.
?Matrix or array without DIM	A matrix or array element was referenced beyond the range of an implicitly dimensioned matrix.
?Maximum memory exceeded	During an OLD operation, the job's private memory size maximum was reached. While running a program, the system required more memory for string or I/O buffer space and the job's private memory size maximum or the system maximum (16K words for BASIC-PLUS) was reached.
?Modifier error	Attempt to use one of the statement modifiers (FOR, WHILE, UNTIL, IF, or UNLESS) incorrectly. An OPEN statement modifier, such as a RECORD-SIZE, CLUSTERSIZE, FILESIZE, or MODE option, is not in the correct order.
?NEXT without FOR	A NEXT statement was encountered in the user program without a previous FOR statement having been seen.
?No logins	Message printed if the system is full and cannot accept additional users or if further logins are disabled by the system manager.
?Not enough available memory	An attempt is made to load a non-privileged compiled program which is too large to run, given the job's private memory size maximum. The program must be made privileged to allow it to expand above a private memory size maximum; or the system manager must increase the job's private memory size maximum to accommodate the program.
?Number is needed	A character string or variable name was used where a number was necessary.
?1 or 2 dimensions only	Attempt was made to dimension a matrix to more than two dimensions.
?ON statement needs GOTO	A statement beginning with ON does not contain a GOTO or GOSUB clause.
Please say HELLO	Message printed by the LOGIN system program. User not logged into the system has typed something other than a legal, logged-out command to the system.
?Please use the RUN command	A transfer of control (as in a GOTO, GOSUB or IF-GOTO statement) cannot be performed from immediate mode.
?PRINT-USING buffer overflow	Format specified contains a field too large to be manipulated by the PRINT-USING statement.
?PRINT-USING format error	An error was made in the construction of the string used to supply the output format in a PRINT-USING statement.
?Program lost-Sorry	A fatal system error has occurred which caused the user program to be lost. This error can indicate hardware problems or use of an improperly compiled program. Consult the system manager or the discussion of such errors in the *RSTS/E System Manager's Guide*.

Message Printed	Meaning
?Redimensioned array	Usage of an array or matrix within the user program has caused BASIC-PLUS to redimension the array implicitly.
?RESUME and no error	A RESUME statement was encountered where no error had occurred to cause a transfer into an error handling routine via the ON ERROR GOTO statement.
?RETURN without GOSUB	RETURN statement encountered in user program without a previous GOSUB statement having been executed.
%SCALE factor interlock	An attempt was made to execute a program or source statement with the current scale factor. The program runs but the system uses the scale factor of the program in memory rather than the current scale factor. Use REPLACE and OLD or recompile the program to run with the current scale factor. (C)
?Statement not found	Reference is made within the program to a line number which is not within the program.
Stop	STOP statement was executed. The user can usually continue program execution by typing CONT and the RETURN key.
?String is needed	A number or variable name was used where a character string was necessary.
?Syntax error	BASIC-PLUS statement was incorrectly formatted.
?Too few arguments	The function has been called with a number of arguments not equal to the number defined for the function.
?Too many arguments	A user-defined function may have up to five arguments.
?Undefined function called	BASIC-PLUS interpreted some statement component as a function call for which there is no defined function (system or user).
?What?	Command or immediate mode statement entered to BASIC-PLUS could not be processed. Illegal verb or improper format error most likely.
?Wrong math package	Program was compiled on a system with either the 2-word or 4-word math package and an attempt is made to run the program on a system with the opposite math package. Recompile the program using the math package of the system on which it will be run.

APPENDIX D
RADIX-50 CHARACTER SET

Many items in RSTS/E, such as file names and extensions, are stored in Radix-50 format. This format allows 3 characters of data to be stored as a 2-byte integer (one 16-bit word). The RAD$ ( ) function converts a Radix-50 word to its 3-character representation. Also, the file name string scan SYS calls convert 3-character strings to Radix-50 format.

The following table shows the complete set of characters capable of being represented in Radix-50 format, their ASCII octal equivalents, and the Radix-50 value by which each character is represented.

Character	ASCII Octal Equivalent	Radix-50 Equivalent (octal)
space	40	0
A-Z	101-132	1-32
$	44	33
.	56	34
unused		35
0-9	60-71	36-47

The value of a character in its 2-byte Radix-50 representation depends on its position. To obtain the octal value of the character in the Radix-50 representation, one must multiply its Radix-50 octal equivalent by the appropriate power of 50 (octal). To gain the full value of the Radix-50 representation of a 3-character string, the values of the 3 characters must be summed. For example, the maximum Radix-50 value (representing the character string 999) is, thus,

$$47*50^2 + 47*50^1 + 47*50^0 = 174777$$

Table D-1 provides a convenient means of translating between the ASCII character set and its Radix-50 equivalents based on position within a string.

A 3-character string is stored left to right in the Radix-50 word. For example, given the ASCII string X2B, the Radix-50 representation is computed as follows.

```
  X  =  113000(octal)
  2  =  002400(octal)
  B  =  000002(octal)
      ---------------
X2B  =  115402(octal)
```

(Note that addition is done in octal.)

To represent a 3-character string in Radix-50 format, the first character of a string (or a single character) is placed in the leftmost position of the Radix-50 word. Thus, for the character X, its representation 30(octal) is multiplied by 50^2 to give 113000(octal), the value shown in Table D-1 for X when it is the first character. The second character in a string is stored in the next position to the right. For the character 2 (in the second position), its representation 40(octal) is multiplied by 50^1 to give 002400, the value shown in Table D-1 for 2 when it is the second character. The third character in a 3-character string is stored in the rightmost position. For the character B (in the third position), its representation is multiplied by 50^0 (which is 1) to give 000002, the value shown in Table D-1 for B when it is the third character. The full octal value of the Radix-50 word is finally gained by adding the value of each character by its position in the string.

Table D-1 Radix-50 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

INDEX

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

RSTS/E Programming
Manual
DEC-11-ORPMA-B-D

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please cut along this line.

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

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 Documentation
146 Main Street ML 5-5/E39
Maynard, Massachusetts 01754