September 1979

This document describes RSTS/E special programming techniques. It includes corrections to previously published material and reflects software enhancements provided by RSTS/E Version V7.0.

RSTS/E Programming Manual

Order No. AA-2726C-TC
Including AD-2726C-T1

SUPERSESSION/UPDATE INFORMATION:	This manual contains information concerning RSTS/E V7.0. It replaces previous versions of the manual.
OPERATING SYSTEM AND VERSION:	RSTS/E V7.0
SOFTWARE VERSION:	RSTS/E V7.0

digital equipment corporation · maynard, massachusetts

1/80-14

Contents

Commercial Engineering Publications typeset this manual using DIGITAL's TMS–11 Text Management System.

2726 ALL

Preface

This manual describes RSTS/E programming techniques and contains information on optimizing the use of the various devices supported on RSTS/E. The techniques described here allow a privileged or knowledgable RSTS/E user full control over input and output operations on specific devices and thus are called device dependent operations.

Chapters 1 through 6 and chapter 10 describe device usage.

Chapter 1 contains information on file structured and non-file structured disks; accounts and disk storage allocation, system files, the concept of privilege, mode specifications, data caching, flexible diskettes, and the disk special function.

Chapter 2 describes the use of magnetic tape; file structured and non-file structured, mode specifications, format labels, the MAGTAPE function, and the magnetic tape special function.

Chapter 3 contains information on line printers; character handling, mode and record values, and error handling.

Chapter 4 describes terminals; record options, multiple terminal service, mode options, pseudo keyboards, and the terminal special function.

Chapter 5 contains information on DECtape; file structured and non-file structured operations.

Chapter 6 describes card readers and paper tape; operations, mode values, and paper tape parity.

Chapter 10 contains information on the DMC device and the DDCMP Network protocol.

Chapter 7 describes the SYS system function calls that allow you to program operations that directly interact with the Monitor, perform special I/O functions, and set terminal and job characteristics. This chapter is useful to both privileged and non-privileged users. Chapter 7 also contains information on the PEEK function.

Chapter 8 describes the interjob send/receive facilities available on the RSTS/E Monitor.

Chapter 9 contains information on programming conventions and techniques, CCL commands, and login.

In addition, this manual contains appendices that describe magnetic tape label formats, card reader codes, error messages and recovery actions, character sets, and a table of handler index codes and Monitor directives.

Conventions

The presentation of material in this manual assumes that you are familiar with the information contained in the *BASIC-PLUS Language Manual* and the *RSTS/E System User's Guide*. The assumption is also made that you have a thorough background in the BASIC-PLUS language.

For information on additional RSTS/E manuals, refer to the *RSTS/E V7.0 Documentation Directory*. Note that all manuals referenced in this text are the most recent version.

All examples in this manual are written to execute in BASIC-PLUS 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. The MFD 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 both generally available and restricted use system programs and message and control files. The system library account 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 a 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 on all User File Directories (accounts) 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 an account and password, the system program LOGIN runs automatically. LOGIN checks the MFD on the system device to determine whether the account number and password given compare with one stored in the MFD. If so, the user is allowed access to the system.

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

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

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

Type	Description	Explanation
Identification	Project-programmer number (account)	Refer to 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.

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 contains UFD information for accounts [0,1] and [1,1] (initially) and its own account and storage information.

The MFD of a public disk, account [1,1], is also used as a UFD when optional software (such as BASIC-PLUS-2 or COBOL) is in use. That is, optional software requires the presence of certain files on the logical device LB:, which is normally defined as SY:[1,1]. To ensure that space is available in the MFD for the creation of user accounts, a cluster size of 16 is recommended for system and public disk MFDs. Also, because the presence of required files in account [1,1] affects the space available for UFDs, you should not assign any user files to that account.

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, the account number is added dynamically into the MFD of that disk and a UFD is created for the user on that disk. A non-privileged user cannot create a file on a private pack unless the non-privileged 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.*

1.1.3 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.* Note that the *RSTS/E System Generation Manual* also describes a BUILD option whereby system programs can be placed in an account(s) other than [1,2].

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.

* 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.

For operational purposes, the system library is accessed automatically during normal system start-up. The system manager specifies the START option when RSTS/E is bootstrapped into memory and, as a result, 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. The last step in the system start-up procedure is to run the ERRCPY program from job 1. Aside from ERRCPY, no other job should run under account [1,2]. The system manager should use other accounts for normal operations.

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

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

Table 1-2: Valid Cluster Size Ranges

Cluster Size	Minimum Size	Maximum Size (decimal)	When Defined
Pack (for any disk)	Device Cluster Size (see Table 1-3)	16	At initialization time by means of DSKINT option (stored in MFD).
Directory (both for MFD and UFD)	Pack Cluster Size	16	At creation of the directory by means of either the DSKINT initialization option, REACT, or SYS system function.
File	Pack Cluster Size	256	At creation of the file by means of either an OPEN or OPEN FOR OUTPUT.

The system manager specifies the cluster size of the disk during disk initialization 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 are 1, 2, 4, 8, or 16, as long as the pack cluster size is equal to or greater than the device cluster size of the disk. Disk device cluster sizes are listed in Table 1-3.

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. For example, one processing consideration is the use of data caching on the system (see Section 1.4.9). While the pack cluster size is set during disk initialization and the cache cluster size can be set and changed during time sharing, the relationship between the two has an effect on the optimal use of the cache. That is, if the pack cluster size and file cluster size are both 4 and you specify a cache cluster size of 8 (see Section 7.2.18), 4 blocks in the cache will contain your file's data and 4 will contain unrelated data. Therefore, if use of data caching is a consideration on your system, the pack cluster size the system manager specifies during disk initialization should be equal to or greater than any cache cluster size you will specify during time sharing.

A directory cluster size is defined for both a Master File Directory (MFD) and User File Directory (UFD) and its minimum value is the pack cluster size. The system manager specifies the MFD cluster size during disk initialization; and specifies the UFD cluster size during account creation. A directory cluster size is equal to or greater than the pack cluster size and 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 (and assumes that no data files are placed in the MFD).

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

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

NOTE

> If account [1,1] is used for the storage of files, such as those required for optional software (BASIC-PLUS-2, COBOL, RMS based languages), the number of accounts that can be created is reduced.

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.)

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

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

1.1.4.2 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 pack 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 cannot 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 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 Executive File — All system Executive (Monitor) code, whether permanently resident in memory or loadable as overlays, 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 loaded 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 system error messages. This file is called ERR.ERR and must exist in account [0,1] on the system 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 fixed or moving head disk. The standard name for this file is ERR.SYS. Optimum efficiency is gained when this code resides on a fast access, fixed head disk. 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 uses the file CRASH.SYS to save 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 between 32 and 80 blocks. To reserve additional, contiguous space, the system manager can create CRASH.SYS at a size larger than required.

1.1.4.7 Run-Time System Files — The account [0,1] on the system disk must contain at least 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 logically 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 nondeletable. 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 (and resident library files) must occupy contiguous space on disk. This condition allows a run-time system (or library) 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 disks, 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 swapping files. The files are numbered 0 through 3. The swapping file numbered 2 is required on all systems; the other files are optional. File 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 files 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 files 0, 1, and 3.

RSTS/E uses the various swapping files in a predefined manner. A highly interactive job which must be removed from memory is stored in the lowest numbered file available. The system searches for an empty space starting at the lowest numbered active file. Conversely, a job with infrequent activity is stored in the highest numbered file 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 file can be either a file or an entire device, for example, a high speed, fixed head disk. The manager can add, as the lowest numbered files, 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 file, 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 file other than file 2 is dynamic. The system manager adds files at the start of time sharing to allow the maximum number of jobs to run. During time sharing, a swapping file can be removed and added again as another device or file. Dynamic addition and removal of swapping files 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 initially 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 can add 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 (ERR=10).

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 that is not associated with 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 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 until the job is logged out or the job changes to a non-privileged account (an account whose project number is not 1).

The NAME AS statement can be used to set the privileged bit in the protection code. 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 protection against the owner's project (<8>) and read protection against all others not in the owner's project (<32>). This protection code combination enables users inside and outside the owner's project to run the privileged program but prohibits those users from reading (and therefore being able to delete or rename) the file.

Note that when a disk file has the privileged <128> bit set, the deletion of that file causes the Monitor to overwrite it with zeroes. The Monitor deletes the file and overwrites it with zeroes when the following conditions are met:

1. the privileged <128> bit is set and
2. you execute an explicit KILL or UNSAVE, or
3. you execute an OPEN FOR OUTPUT with the same filename implicitly deleting the file, or
4. the file is tentative (see Section 1.4.6 - MODE 32%) and you execute a reset CLOSE with a negative channel number, or
5. you perform a ZERO operation on the file's account.

Because the Monitor is in a locked state during the overwrite operation, no other programs on the system can perform file processing. Therefore, permanent privilege is required to set the <128> bit. Also, because a file with the <128> bit set in the protection code can be overwritten with zeroes, you should ensure that sensitive data files (such as ACCT.SYS) are protected from inadvertent zeroing. That is, rename the file to its required protection code plus <128> but do not set the <64> compiled bit. Note that to process a privileged (<128>) file through the QUE system program, you must have permanent privilege. Also, an attempt to delete such a file with the QUE program /DE switch will fail.

The NAME AS statement also can 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 program, or compiled 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 non-privileged 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 unless it changes to a non-privileged account.

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. For example, 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.

Note that to prevent deliberate or inadvertent misuse of privilege, DIGITAL recommends that user-written privileged programs initially drop privilege and then regain privilege only as needed.

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.

The system manager should take steps to prevent unauthorized access to privileged accounts. For example; keep privileged account passwords confidential and change them from time to time, and do not use passwords that are obvious (such as a person's name). The responsibility of maintaining system integrity extends to other privileged users because 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 non-privileged users in the same way standard system library programs do. For example, the program TTYSET allows a user to change the characteristics of his terminal. Such an operation is privileged. With temporary privilege, however, TTYSET executes the normally privileged operation for an owner of a non-privileged account. TTYSET does not generate the expected protection violation error.

The same TTYSET program additionally allows a privileged user to change the characteristics of another user's terminal. But TTYSET is coded to ensure that a user attempting to change the characteristics of another's terminal is indeed a permanently privileged user. As a result, TTYSET makes available to a non-privileged 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 you to process non-RSTS/E file structured disks under RSTS/E and to use an entire disk as a single file. It also allows system programs, such as SAVE/RESTORE (see the *RSTS/E System Manager's Guide*) to optimally process file structured disks.

1.3.1 Opening a Disk for Non-File Structured Processing

To initiate non-file 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 "DL1:" FOR INPUT AS FILE 1%
```

```
100 OPEN "DL1:" AS FILE 1%
```

Both allow reading and writing of physical blocks on RL unit 1. A third sample statement results in the error ?DISK PACK IS NOT MOUNTED (ERR=21).

```
100 OPEN "DL1:" FOR OUTPUT AS FILE 1%
```

To prevent other programs from accessing a non-file 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 understand 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 non-file structured I/O operation. Device clusters are numbered from 0 to the maximum shown in Table 1-3.

Table 1–3: Non-File Structured Disk Default Characteristics

Device	Device Cluster Size	Default Buffer Size (in Bytes)	Maximum BLOCK Number
RF11	1	512	(number of platters*1024)-1
RS03	1	512	1023
RS04	1	512	2047
RK05,RK05F	1	512	4799
RL01	1	512	10219
RL02	1	512	20459
RK06	1	512	27103
RK07	1	512	53767
RP02	2	1024	19999
RP03	2	1024	39999
RM02,RM03	4	2048	32919
RP04,RP05	4	2048	41799
RP06	8	4096	41799

After the user program opens a disk device for non-file structured processing, the program uses the BLOCK option in GET and PUT statements to read and write specific logical blocks on the disk. The number that the user specifies designates a device cluster number. Thus, on an RK05, BLOCK 4100% refers to logical block 4100 on the disk, because the device cluster size for an RK05 is 1. On an RP03, BLOCK 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 non-file 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%, BLOCK 39999.
```

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

You 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, include 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 non-file 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 BLOCK 0 or omit the BLOCK option. Once the disk has been accessed, omitting the BLOCK option or specifying BLOCK 0 in a GET or PUT statement accesses the next sequential device cluster. Note that you can use RECORDSIZE to read a partial block (see Section 1.4.17), however the system positions itself at the start of the next cluster following the operation.

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 non-file structured access. The statement:

```
100 OPEN "DK1:" AS FILE 1%\ GET #1%, BLOCK 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 (ERR=11) error occurs.

1.3.2.1 Non-File Structured Block Access - MODE 128%. — If your system is configured for large files (greater than 65535 blocks), you can specify MODE 128% in a non-file structured OPEN statement to access disk blocks. If your system does not have large file capability, or if MODE 128% is not specified, a non-file structured disk OPEN statement allows access to device clusters.

When you specify MODE 128%, you can perform read/write operations on individual disk blocks. To access blocks on the disk, specify MODE 128% in the OPEN and use the BASIC-PLUS modifier BLOCK in the GET or PUT statement. The BLOCK modifier accepts a floating-point argument that represents the desired block (where block 1 is the first block in the MFD.) The use of the BLOCK modifier in GET and PUT statements is described in the *BASIC-PLUS Language Manual.*

If your system does not support large files, a MODE 128% specification returns the ?MISSING SPECIAL FEATURE error (ERR=66).

1.3.3 Privilege and Access

A disk that is being processed in non-file structured mode need not be logically mounted. After a user inserts the disk into its drive, any user may read or write it. A non-privileged user cannot read or write the disk in non-file structured mode while another user is accessing it. Such an attempt results in a ?PROTECTION VIOLATION (ERR = 10) for the non-privileged 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 non-file structured processing. In this case, a non-privileged 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 accesses it has. The STATUS variable is discussed in the *BASIC-PLUS Language Manual.*

1.3.4 Assigning a Disk Unit

A privileged user may assign a dismounted disk unit to the current job. This action prevents access by other users to the drive when the privileged user wishes the program to perform non-file structured operations on a volume mounted in the drive.

When a dismounted disk is assigned, the system limits access to the drive. The drive cannot be logically mounted. The job to which the drive is assigned can gain both read and write access to the disk. If a non-privileged user attempts to access the disk, the ?PROTECTION VIOLATION error is generated. Privileged users can read the disk in non-file structured mode but cannot write on the disk.

The output of the SYSTAT disk status reports show an assigned drive as non-file structured (NFS) and private (PRI). The SYSTAT report is in the following format:

```
Disk structure

Disk     Open     Free     Cluster     Errors     Name     Comments
DS0       3        75         1           0       SWAP0    Pri
DS1       1                   1           0                Pri,NFS
```

1.4 File Structured Disk Operation

In file structured disk operation, the user organizes data into files. The system manager uses the DSKINT option during system initialization to set up a skeletal file structure on a RSTS/E disk. During time sharing, the user can create files with the OPEN and OPEN FOR OUTPUT commands, which are described in the *BASIC-PLUS Language Manual.* That manual also 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 "FILE.DAT" AS FILE N%, MODE M%
```

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

Table 1-4: MODE Specifications for Disk Files

MODE	Meaning
0%	Normal read/write.
1%	UPDATE mode.
2%	APPEND to file.
5%	Guarded UPDATE (4%+1%).
8%	Special extend.
16%	Create contiguous file.
32%	Create tentative file.
64%	Create contiguous file conditionally.
128%	No supersede.
256%	User data caching.
512%	Create file - place at beginning of directory (with 1024%).
1024%	Create file - place at end of directory.
2048%	Sequential data caching (with 256%).
4096%	Read normally regardless (privileged).
8192%	OPEN file 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 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 and write 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 1.4.12. 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.*

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

The UNLOCK statement has the form:

```
line number UNLOCK <expression>
```

where <expression> is the internal channel number of the file that is opened for update.

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 31 blocks in the buffer to be locked in this manner and allows up to seven locks on the file (see Section 1.7). The same rules for a single locked block are applicable for the range of locked blocks.

If your system is generated for the large file option, you can OPEN the file in UPDATE mode (1% or 5%) and extend it beyond the current end-of-file. To extend the file, use the following procedure:

1. OPEN the file for UPDATE mode.
2. GET block 1 (the first block of the file).
3. Use the SPEC% function (see Section 1.7) to place an explicit lock on block 1.
4. Extend the file to the desired length beyond the current end-of-file with PUT statements.
5. Unlock block 1 (see Section 1.7).

* Executing an END or CHAIN statement or executing the last statement of the program implicitly closes all files.

The extended blocks are now available to users of the file. Note that this extend operation is legal only on systems which have the large file option enabled. If this option is not present on the system, the extend operation will fail.

1.4.2.3 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%.

Note that if your system is generated for the large file option, you can OPEN the file in UPDATE mode and extend it beyond the current end of the file. The extend procedure is described in Section 1.4.2.2.

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, specify MODE 2% in the OPEN statement. Do not use the OPEN FOR OUTPUT statement, as it deletes the existing file. Specify MODE 2% only with Record I/O files. For example:

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

The system opens the file DATA.DAT under the current account on the system disk. The next output operation creates a new block and appends it to the last block in the file that contains data. 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 BLOCK number. When the PUT statement does not include the BLOCK option, the system writes the next sequential block.

The following sample program illustrates APPEND by showing its use in a classroom environment. Each student enters 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$(10%)
        \OPEN "SCIENC.EXP" AS FILE 1%, MODE 2%
        \IF (STATUS AND 1024%) THEN
                PRINT "WRITE ACCESS NOT GRANTED."
                \PRINT "TRY AGAIN IN A FEW MINUTES."
                \GOTO 800
400     FIELD #1%, 8%*I% AS B$, 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. However, the size of the file open on each channel is only current as of the last time the file was closed or as of the last time directory information was updated on disk with MODE 8% (see Section 1.4.4). Note that if the system is generated for large file support, the most current file size information is available to all channels on which a file is open. Thus, when a program opens the same file on more than one channel and the system does not support large files, the 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. When a program opens the same file on more than one channel and the system does support large files, special processing of error number 11 is not required. Because the current file size is available on large file systems, the occurrence of error number 11 always indicates the end of the file.

1.4.4 Special Mode for Extending Files - MODE 8%

To force RSTS/E to update a file's size data and retrieval pointers 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 and the retrieval pointers for each allocated cluster.

The following sample statement shows the MODE 8% option.

```
10 OPEN 'DATA.DAT' 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 physically 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). Note that you can specify MODE 64% (see Section 1.4.7) to create a contiguous file conditionally. That is, the file is made contiguous if possible, otherwise, it is made non-contiguous and no error is returned.

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 Creating a Tentative File - MODE 32%

Specify MODE 32% in the OPEN FOR OUTPUT statement to create a file which does not become permanent until it is closed. When used, the file that you create is not made permanent until you execute a CLOSE on that file. If a file of the same name currently exists, it is not superseded until you close the tentative file.

When you create a tentative file, the system searches for an existing file of the same name. If the disk on which the file is created is public, the public structure is searched. If a file of the same name is located and its protection code does not allow deletion, a ?PROTECTION VIOLATION error (ERR=10) is returned. If a file of the same name is located and it can be deleted, it is left intact (not deleted) until a CLOSE on the tentative file is executed.

A successful OPEN statement causes an entry for the tentative file to be made in the directory. The entry marks the tentative file for deletion. Should the system crash or the job reset the channel (with a negative channel number in the CLOSE statement) prior to closing the file, the tentative file is deleted. Note that tentative file directory entries only appear on a directory listing that contains files marked for deletion.

When you close a tentative file, the system again searches for a file of the same name. If such a file is found and it can be deleted, the system deletes it and makes the tentative file permanent. If a file of the same name is found and its protection code does not allow deletion, a ?PROTECTION VIOLATION error occurs. However, the system closes the tentative file and renames it as follows:

    TM?nnn.TMP

where:

?	is an alphabetic indication of the file's channel (A=0, B=1, C=2, etc.).
nnn	is the job number.

Note that this operation can cause multiple copies of this name to exist in a directory.

1.4.7 Creating a Contiguous File Conditionally - MODE 64%

Specify MODE 64% in the OPEN FOR OUPUT statement to create a contiguous file if contiguous space is available for it. When used, it causes the Monitor to create a contiguous file based on the following conditions:

1. If there is enough contiguous space available on the disk to contain the file, the Monitor creates a contiguous file.
2. If there is not enough contiguous space on the disk to contain the file, the Monitor creates a non-contiguous file. No error is generated.

Note that if MODE 16% is also set for the file (see Section 1.4.5), MODE 64% is ignored.

1.4.8 No Supersede - MODE 128%

Specify MODE 128% in the OPEN FOR OUTPUT statement to create a file which will not supersede an existing file of the same name. MODE 128% notifies the Monitor that, if a file of the same name currently exists, the existing file should not be deleted. That is, if the file already exists, a ?NAME OR ACCOUNT NOW EXISTS error (ERR=16) is returned.

1.4.9 Data Caching

When a user job executes a read request, the Monitor performs a disk access and transfers the requested data from the disk to the user job's buffer. On systems that have many jobs requiring large amounts of data, the equally large number of disk data transfers can have an impact on response time.

Data caching is a method whereby a privileged user can reduce data transfers from disk. When caching is enabled, the Monitor stores the most recently read (accessed) data blocks in an area of memory called the cache. If the user job requests a data block that is present in the cache, the Monitor copies the requested data directly from the cache into the job's buffer. Thus, the Monitor avoids a physical disk access operation.

Data caching is most useful for read operations in that it can minimize disk transfers. In a write operation that modifies existing data, the data is updated (whether in the cache or on disk) but no new data is installed in the cache.

The system manager installs caching on the system and optionally sets its parameters during system initialization. A privileged user enables or disables caching and determines the size of the cache by means of the disk cache enable SYS call (see Section 7.2.18) or the UTILTY system program (see the *RSTS/E System Manager's Guide*). The SYS call or UTILTY are also used to specify caching for a particular file by marking the file's UFD entry (see Section 7.2.18). If a new file is to be created and accesses to it are to be cached, use the MODE values in the OPEN statement as described in Sections 1.4.9.1 and 1.4.9.2. However, because the use of caching MODE values is privileged, a privileged user should designate a file for caching by means of the file's UFD entry (i.e., use the SYS call or UTILTY). Once a file's UFD is marked for caching, it will be cached on OPEN regardless of the user's privilege as long as caching is enabled on the system.

When caching is implemented, the cache receives all data transfer requests that are directed to the disk driver. Because the cache contains data, read operations on data that is in the cache occur without placing a load on the disk driver. The Monitor constantly updates the cache such that it contains the most recently requested data.

During system initialization (or with the disk cache enable SYS call), the system manager sets the cache cluster size. The cluster size (1, 2, 4, or 8 blocks) determines the amount of data that can be present in the cache at any particular time and, in effect, the number of read requests that can be resolved in the cache before access to the disk driver is required. For example, when the cache cluster size is set at 8 blocks, any read operation that installs data in the cache causes 8 physically contiguous blocks (including the requested blocks) to be installed. Note that the cache cluster size should not usually be greater than the pack cluster size that was set during disk initialization (see Section 1.1.4.1). The Monitor allocates space for the cache from the Extended Buffer Pool (XBUF), see Section 7.2.18.

1.4.9.1 User Data Caching (Random Mode) - MODE 256% — A privileged user can specify MODE 256% in an OPEN statement to cause data transfers to and from a file to be cached. MODE 256% has effect only if data caching is enabled on the system (see Section 7.2.18).

The Monitor schedules a job's data transfers into the cache on the basis of time since last access. That is, new data replaces data currently in the cache if the currently cached data:

1. is the data with the longest time since last access, and
2. has been in the cache for more than the minimum residency as established by the system manager.

If a data block meets these requirements, it is replaced when the Monitor seeks to place a new data block in the cache.

As an example of random cache operation, consider a GET statement that is executed on a file opened with MODE 256%. When a read on the cached file occurs, the Monitor examines the content of the cache to determine if the requested data item is present. If the data is in the cache, the data is copied from the cache buffer that contains it. The data is made available to the program and the cache buffer is linked to the beginning of the list of cache buffers. The list of cache buffers is maintained in order of increasing time since last access.

If the requested data item is not present in the cache, the Monitor examines the list of cache buffers to determine the time of last access for the oldest item in the cache. If the time is less than the minimum residency, the requested data cannot be installed in the cache and a normal disk read is automatically performed. If the time is greater than the minimum residency, the requested data is installed in that cache buffer and made available to the program.

1.4.9.2 User Data Caching(Sequential Mode) - MODE 2048% — MODE 2048% has effect only if the file is being cached. That is, MODE 256% is set, or the file's UFD entry is marked for caching, or caching is set for all data on the system (see Section 7.2.18). Note that sequential mode data caching requires a cluster size greater than 1, although no error is returned if the cluster size is 1.

When MODE 2048% is specified in an OPEN statement, data blocks from that file placed in the cache as a result of that OPEN are not subject to the minimum residency. However, MODE 2048% overrides the minimum residency only for that file's data blocks, not for the entire cache. That is, when data blocks from a sequentially cached file are placed in the cache, a read on the last block of the cluster causes it to become the "oldest" data block in the cache. It is the first to be replaced when the Monitor seeks to place new data blocks in the cache. MODE 2048% has the effect of speeding the replacement process in the cache and thus minimizes the space required by the cache.

As an example of sequential cache operation, consider a GET statement that is executed on a file opened with MODE 2304% (256%+2048%). When a read on the cached file occurs, the Monitor examines the content of the cache to determine if the requested data item is present. If the data is in the cache, the data is copied from the cache buffer that contains it. The data is made available to the program. If the data is in the last block of a cache cluster, that cluster is made available for replacement in the cache.

When a read operation is performed on the first block of a cache cluster, the Monitor installs a full cluster of data in the cache. That is, if the system manager sets a cluster size of 8 for the cache, the requested data block plus the next 7 blocks are read from the disk into the cache. Thus, the next 7 reads are present in the cache. When the last block of the cache buffer is read, the buffer is immediately made available for the installation of new data as requested by other read operations.

If the GET statement causes more than one cache cluster to be read, all of the requested data blocks are made available to the program but only the last cache cluster is installed. However, if the last data block read is the last block in a cache cluster, no data is installed in the cache. Thus, if the cache cluster size is defined as 1 and sequential mode is specified, no data blocks will be installed in the cache (i.e., every data block is the last block in a cache cluster).

1.4.10 Creating and Placing a File at the End of the Directory - MODE 1024%

You specify MODE 1024% to override the pack default and specifically place the file at the end of the directory. This file placement is useful for those files that are infrequently accessed or are not time-critical. Because the Monitor always searches for files starting at the beginning of the directory, placing non-critical files at the end speeds access to the first part of the directory.

MODE 1024% is specified in the OPEN FOR OUTPUT statement and causes the Monitor to place the file at the end of the current account's directory. If you do not specify MODE 1024%, the Monitor places the file in the directory as directed by the pack default condition. This condition depends on the system manager's response to the NEW FILES FIRST query during disk initialization. Therefore, in the absence of MODE 1024%, file directory placement is a function of the disk pack's default condition. For example, if you create the file on DB1: and do not specify MODE 1024%, the Monitor uses the DB1: default to place the file. If the device is part of the multiple-disk public structure (SY:), the Monitor selects the disk pack with the most free space and uses that pack's default.

1.4.11 Creating and Placing a File at the Beginning of the Directory - MODE 1536%

MODE 1536% (MODE 1024% + MODE 512%) is specified in the OPEN FOR OUTPUT statement and causes the Monitor to place the file at the beginning of the current account's directory. If you do not specify MODE 1536%, the Monitor places the file in the directory as directed by the pack default condition. This condition depends on the system manager's response to the NEW FILES FIRST query during disk initialization. Therefore, in the absence of MODE 1536%, file directory placement is a function of the disk pack's default condition. For example, if you create the file on DB1: and do not specify MODE 1536%, the Monitor uses the DB1: default to place the file. If the device is part of the multiple-disk public structure (SY:), the Monitor selects the disk pack with the most free space and uses that pack's default.

You specify MODE 1536% to override the pack default and specifically place the file at the beginning of the directory. This file placement is useful for those files that are frequently accessed. For example, on most RSTS/E systems it is desirable to place $LOGIN in the first position of the system disk's directory. Because $LOGIN is constantly being opened for access to log jobs into the system, its placement as the first file in the directory eliminates lengthy searches through the system library directory.

1.4.12 Reading a File During Processing - 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 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. When a file is opened using MODE 4096%, other users can open the file in update mode.

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 (ERR=19) is not generated when the program reads a block being updated (although that block may contain inconsistent data).

MODE 4096% requires privileges because of the danger involved in reading data which is subject to change.

1.4.13 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 gain read access to a data file when write access is not desired, specify the MODE 8192% option in the OPEN FOR INPUT statement. Write access is never granted when a file is opened for MODE 8192%. Note that you cannot use MODE 8192% to open a file that is currently opened for update (MODE 1%). However, you can use MODE 8192% on files that are opened normally (MODE 0%). If a file is opened for update and you specify MODE 8192%+1%, the file is opened without write access. If the file is not yet opened and you specify MODE 8192%+1%, subsequent opens on that file must be made with MODE 1%.

The following statement illustrates the procedure used to specify MODE 8192%:

```
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).

1.4.14 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.

1.4.15 Simultaneous Disk Access

RSTS/E permits several users to read from the same file simultaneously, but only one user can write into a 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, users may fail to obtain write privileges to a file that is not write-protected against them. 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.* Attempting to do so results in a ?PROTECTION VIOLATION. 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 the *BASIC-PLUS Language Manual.* The example given in Section 1.4.3 performs such a check.

1.4.16 Disk Optimization

Whenever you open a file on the public structure, the system searches the directories of all public disks. This search determines whether the file exists. To avoid the overhead of searching multiple directories, put 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 you find this impractical and must store more than one such file on one private disk, dedicate an entire account to each file. This arrangement reduces directory search overhead.

However, if you must save more than one file under an account, create the more frequently accessed ones first (or use MODE 1536%, see Section 1.4.11) to ensure better directory organization. If you 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.* 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, 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, you can pre-extend a disk file to its maximum length when you create it and can specify that contiguous space be used. Pre-extension reduces directory fragmentation. Contiguous space reduces window turning, which is the process of following UFD retrieval pointers to locate a specific block within a file.

* A file may, however, be open simultaneously in update mode and read during processing mode as described in Section 1.4.12.

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.

1.4.17 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 BLOCK may be used. For example, with a buffer of 520 bytes, BLOCK 1 retrieves the first block and 8 bytes of the second block. BLOCK 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, the buffer size (RECORDSIZE) must be a multiple of 512 bytes.

1.5 Flexible Diskettes

The RX11/RX01 and RX211/RX02 Flexible Diskettes (sometimes referred to as floppy disks) can be used only as non-file structured devices.* The device name for the flexible diskette is DX. The current flexible diskette implementation allows only a single .BAS file to be stored on diskette unit 1, for example, with the SAVE command:

```
SAVE DX1:
```

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

```
OLD DX1:
```

and

```
RUN DX1:
```

* The FLINT and FIT system programs are available to allow users to transfer specially formatted data to the RSTS/E environment (see the *RSTS/E System User's Guide*).

A flexible diskette is divided into 77 tracks (numbered 0 through 76), each of which consists of 26 sectors (numbered 1 through 26). Consequently, there are 2002 records (numbered 0 through 2001) and each record is 128 bytes for RX01 and single density RX02 or 256 bytes for double density RX02 on each diskette.

A flexible diskette can be opened and accessed in either of two modes, as summarized in Table 1-5.

Table 1-5: MODE Specifications for Flexible Diskette

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 for RX01 and 256 bytes for RX02. Open the diskette 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 single density or a 256-byte double density record from the diskette. The RECORD option, if present, defines a specific record on the diskette. If the RECORD option is omitted or RECORD 0% is included, the next sequential record is read.

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

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

If -32768% (formed by 32767% + 1%) is included in the RECORD option (e.g., RECORD N%+32767%+1%), sectors are interleaved according to the algorithm discussed in Section 1.5.2.

A PUT statement writes a 128-byte single density or a 256-byte double density record on the diskette.

```
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 (see Section 1.5.3) with each of the records. C% must be a positive non-zero number.

NOTE

If a single density diskette is inserted into an RX02 drive, the buffersize on a sector mode open is 256 bytes (the length of two sectors). Thus, the statement, GET RECORD N%, reads record N% and record N%+1%. To ensure that you read one record, include COUNT 128% in the GET statement.

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 flexible diskette (e.g., RSX-11M, RT11). Track 0 is unavailable in order to reserve its use for IBM-compatible labels.

The following statement opens the diskette 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 diskette. 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 diskette can be accessed only by the first GET statement after the device is opened.)

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

```
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 positive non-zero number.

Block mode operations can be performed in sector mode by opening the diskette 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.

1.5.2.1 Flexible Diskette RECORD Modifiers — When you perform I/O operations on flexible diskettes, you can use the RECORD option in GET and PUT statements to modify the actions of the diskette drive. The RECORD modifiers and the actions they cause are as follows.

RECORD 8192% allows you to access logical record zero on the flexible diskette. Under normal operation, access to logical record zero is not allowed after the first I/O operation is performed. However, the statement:

```
GET #n, RECORD 8192%
```

accesses logical record zero providing that the condition (RECORD% AND 8191%)=0% is true.

RECORD 16384% is used in the PUT statement and allows you to write a Deleted Data Mark to the diskette (see Section 1.5.3).

RECORD 32767%+1% causes the following I/O operation to be performed in block mode. That is, when block mode is desired on a diskette that is open in sector mode (MODE 16384%), you can specify RECORD 32767%+1% in the GET or PUT statement. With RECORD 32767%+1%, the I/O operation you perform is done in block mode.

1.5.3 Deleted Data Marks

Each sector of a flexible diskette contains a bit called the Deleted Data Mark, in addition to its data. When an INPUT or GET operation from the diskette 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 one sector, the last sector read into the buffer is 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 Flexible Diskettes

The RECORDSIZE option in the OPEN statement on a flexible diskette can specify a value which is not a multiple of the default buffer size (512 bytes in block mode and 128 bytes or 256 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 legal value 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.

NOTE

> When the COUNT option is used in a GET statement, the COUNT argument must be a positive even number. If an odd number (or 0) appears in the COUNT, an ?ILLEGAL BYTE COUNT FOR I/O error (ERR=31) is returned.

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.

1.5.5 Flexible Diskette Special Function - SPEC%

The SPEC% function performs special operations on flexible diskettes, disks (see Section 1.7), magnetic tape (see Section 2.12), and terminals (see Section 4.6).

For flexible diskette devices, the SPEC% function allows you to obtain the density (single or double) of the current diskette, to mount a new diskette and recompute the density, and to reformat an RX02 diskette for a desired density. Because the RX02 flexible diskette drive supports single and double density diskettes, the SPEC% function is especially useful for programmed diskette operations. For example, SPEC% allows you to mount a series of single and double density diskettes without having to close and reopen the device for each mount. That is, the driver computes density once; during the initial open. If you insert a second diskette that is incompatible with the initially computed density, read or write operations will fail. SPEC% permits you to include an instruction in your program that causes the driver to recompute the density. Also, for RX02 flexible diskette drives, SPEC% permits you to specify a density reformat operation.

The SPEC% function for flexible diskettes has the following format:

```
VALUE%=SPEC%(FUNCTION%,PARAMETER,CHANNEL%,HANDLER INDEX%)
```

where:

VALUE%	depends on the particular function code you specify in FUNCTION%.
FUNCTION%	is a function code that specifies the desired operation. The codes are as follows:

FUNCTION%=0%	returns the density of the currently mounted diskette in the form: DENSITY%=VALUE% AND 255%. If DENSITY%=1%, the diskette is single density; if DENSITY%=2%, the diskette is double density. Note that PARAMETER must also be 0.
FUNCTION%=1%	causes the diskette driver to recompute density. If the diskette has been changed in the drive without closing and reopening the I/O channel, issue this code prior to any I/O operation on the diskette. This function also returns the computed density as described in FUNCTION%=0%. Note that PARAMETER must be 0.
FUNCTION%=2%	reformats the current diskette to the density specified in PARAMETER. PARAMETER equals 1 for single density and 2 for double density. Note that this operation is allowed only on RX02 drives and that any data on the diskette prior to the operation is lost.

PARAMETER	see the description of FUNCTION%.
CHANNEL%	is the I/O channel on which the operation is to be performed.
HANDLER INDEX%	the handler index on the I/O channel open on CHANNEL%. The index for flexible diskettes is 18%.

SPEC% can require as much as 20 seconds to reformat the density of an RX02 diskette and cannot be interrupted with CTRL/C. Note that if the operation is interrupted (by power failure or catastrophic error), the diskette is rendered unusable. That is, the diskette will contain both single and double density. To recover, you must reformat the diskette.

The following errors are possible during a SPEC% operation.

Meaning	Error
?DEVICE HUNG OR WRITE LOCKED A hardware error occurred. This can often be a transient condition. Retry the operation.	14
?MISSING SPECIAL FEATURE An attempt was made to reformat on an RX01 flexible diskette drive. The use of SPEC% to reformat diskette density is allowed only on RX02 drives.	66

SPEC% can be used in flexible diskette programming operations to ensure that sector opens are correctly handled. That is, the conflict between 128-byte single density buffersizes and 256-byte double density buffersizes can be resolved by the following method:

To field the buffer;

```
FIELD #channel number, 128%*DENSITY% AS BUFFER.RX02$
```

To PUT the buffer;

```
PUT #channel number, COUNT 128%*DENSITY%
```

where DENSITY% is;

```
DENSITY%=SPEC%(0%, 0, CHANNEL%, 18%) AND 255%
```

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.

1.7 Disk Special Function - SPEC%

The SPEC% function performs special operations on disk, flexible diskettes (see Section 1.5.5), magnetic tape (see Section 2.12), and terminals (see Section 4.6).

For disk devices, the SPEC% function allows you to explicitly lock a maximum of seven disk block ranges on a file that is open for update (MODE 1% or MODE 1%+4%, see Section 1.4.2). A locked range (from 1 to 31 blocks) is one that can not be accessed by another user or from another channel. Thus, SPEC% extends the use of guarded update, which locks the last block or blocks read on a file.

SPEC% also allows you to release explicit or implicit locks. Note that when you close a file, all explicit and implicit locks are released for that file.

The SPEC% function for disk files has the following format:

```
VALUE%=SPEC%(FUNCTION%, BLOCK, CHANNEL%, HANDLER INDEX%)
```

where:

VALUE%	depends on the particular function code you specify in FUNCTION%. In most cases, VALUE% is equal to the BLOCK parameter.
FUNCTION%	is a function code that specifies the desired operation. During normal I/O operations, a block, or range of blocks, is implicitly locked when you read the file with a BASIC-PLUS GET statement. The SPEC% function allows you to convert implicit locks to explicit locks and to release selected locked blocks. The code specified in FUNCTION% determines the use of SPEC%. The codes are as follows:

FUNCTION%=0	release all locked blocks.
FUNCTION%=1	release the current implicit lock.
FUNCTION%=2	convert the current implicit lock to an explicit lock.
FUNCTION%=3	release the explicitly locked block specified in the BLOCK parameter. If BLOCK is 0, all explicitly locked blocks are released. However, implicitly locked blocks remain locked.
FUNCTION%=4	convert an implicit lock to an explicit lock and release the implicit lock.

BLOCK	specifies the starting block number for releasing an explicit lock. Note that BLOCK must be a floating-point number.
CHANNEL%	is the I/O channel on which the operation is to be performed.
HANDLER INDEX	the handler index of the I/O device open on CHANNEL%. The index for disk devices is 0%.

If you open a file with a RECORDSIZE greater than 512, SPEC% allows you to lock more than one block when you GET a range of blocks read into the buffer. For example, if you open the file with RECORDSIZE 1024%, each GET operation reads (and implicitly locks) two blocks. If you explicitly lock blocks 2 and 3 as follows:

```
GET#1%, RECORD 2%
VALUE%=SPEC%(2%,0,1%,0%)
```

you can then read blocks 3 and 4 (GET RECORD 3%) and cause implicit locks on these blocks. Note that if you attempt to lock a range of blocks that overlap an already explicitly locked range, the Monitor returns the ?DISK BLOCK IS INTERLOCKED error. Also, if a range of blocks is locked, an explicit release of those blocks must refer to the first block in the range.

The following errors are possible during a SPEC% operation:

Meaning	Error
?NO ROOM FOR USER ON DEVICE There are too many locks pending on this device. You can lock a maximum of seven ranges of blocks on a file.	4
?CAN'T FIND FILE OR ACCOUNT You specified function code 3 in the FUNCTION% parameter and attempted to unlock a block that was not locked.	5
?PROTECTION VIOLATION You attempted to explicitly lock a block that had not been implicitly locked. An attempt to lock a block after a PUT or UNLOCK can cause this error.	10
?DISK BLOCK IS INTERLOCKED You attempted to explicitly lock a range of blocks that overlaps an already explicitly locked range of blocks.	19

Chapter 2 Magnetic Tape

Magnetic tape I/O is processed under RSTS/E in one of two forms: file structured magnetic tape and non-file structured magnetic tape. In addition to this distinction, RSTS/E supports two magnetic tape file labels; ANSI and DOS.

NOTE

> Where ANSI is used in RSTS/E documentation, it refers to the RSTS/E implementation of American National Standard X3.27–1978 - magnetic tape labels and file structure for information interchange. RSTS/E implements a subset of this standard. Also, RSTS/E uses U (undefined) record format, which is not defined in ANSI standard X3.27–1978

A magnetic tape must not be written with both ANSI and DOS file labels. Illogical interpretations result if these labels appear on the same tape. Also, RSTS/E allows you to specify the content of the ANSI label written to the tape (see Table 2–4). However, to read or write ANSI labeled tapes, you must format the data (see Section 2.2.7) or use the PIP utility program described in the *RSTS/E System User's Guide*.

A user program controls file structured magnetic tape 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 the *BASIC-PLUS Language Manual*. It also controls the system action when the file is subsequently closed.

By default, RSTS/E writes tape records of 512 bytes. Table 2–1 lists standard system defaults for magnetic tape.

Table 2-1: System Parity and Density Defaults* for Magnetic Tape

TU10 7-Track	TE10 TU10 TS03 9-Track	TE16 TU16 TU45 TU77 9-Track	TS11 9-Track
800 bpi ODD parity DUMP mode	800 bpi ODD parity	800 bpi ODD parity	1600 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 or SPEC% functions for both non-file structured and file structured and the MODE option for non-file structured) described in this chapter.

The following sections describe magnetic tape 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 magnetic tape operation on a CLOSE statement. The remainder of this chapter discusses non-file structured magnetic tape operations, the MAGTAPE function, the magnetic tape special function (SPEC%), and error handling techniques.

2.1 The File Structured Magnetic Tape OPEN FOR INPUT

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

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

The OPEN FOR INPUT statement searches for a specified file on a designated tape 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 magnetic tape permits read access only.)

In the above example, the system associates tape 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. Note that account numbers are ignored on ANSI labeled tapes.

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.

* An optional feature patch may be installed to change system tape parity and density defaults. Consult the *RSTS/E V7.0 Release Notes* for details.

Table 2-2: Magnetic Tape OPEN FOR INPUT MODE Values

MODE	Meaning
0%	Read file label 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 file label.
24576%	Search for an ANSI formatted file label.

If the file is found, it is opened 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 a DOS file structured magnetic tape under an account other than the current account, supply the proper project-programmer number in the OPEN statement. Project-programmer numbers are ignored on ANSI labeled tape. Because the system does not check protection codes for files on tape, any user can access a magnetic tape file.

Under DOS file structured operations, the system reads magnetic tape records into a 512-byte buffer. In certain cases, however, you 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 tape unit 0 for input and allocates M% bytes of buffer space for data transfer operations.

For ANSI labeled tapes, 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. To avoid any loss of data, you should ensure that the block length is an even value.

2.1.1 Searching for a Label on INPUT

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 or the tape is not properly positioned, 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 for the desired file 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 label record, the system automatically rewinds the tape to the first file label record and begins reading labels file by file.

To override this automatic rewind feature, include MODE 2% in the OPEN statement. In this case, the system reads the tape from its current position and, if no match occurs, continues reading file 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

MODE 16384% in the OPEN FOR INPUT statement causes the system to search for the magnetic tape 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.

MODE 24576% (16384% + 8192%) in the OPEN FOR INPUT statement causes the system to search for the magnetic tape file whose filename is specified. In this case, the tape label must be written in ANSI format for the search to be considered successful.

When you omit MODE 16384% or 24576%, the system assumes label records on the tape, either DOS or ANSI, are in the default format as specified by the system manager when time-sharing operations began or as specified by the user when the unit was reserved to the current job (ASSIGN, MOUNT or the assign SYS function call). The MODE value overrides any current defaults for labeling.

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.

2.1.4 The GET Statement

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

```
100   GET #N%
```

N% is the channel on which the device is open. This statement reads the next sequential block in the file. For DOS format tapes, the buffer is 512 bytes long unless you specify a larger buffer with the RECORDSIZE option when the file is opened. For ANSI labeled tapes, the buffer size is the block length read from the header 2 label (HDR2). Magnetic tape 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 the *BASIC-PLUS Language Manual.* Attempting to read beyond the end of the file results in ?END OF FILE ON DEVICE (ERR = 11).

Note that in processing ANSI labeled tape, RSTS/E does not convert the data. The user program must interpret data retrieved with the GET statement.

2.1.5 Example of OPEN FOR INPUT Statement

The values for MODE discussed in the previous sections 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:

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

This statement opens the file MARKIE on tape unit 1 and associates it with channel 3%. MODE 24772% is the sum of MODE 32% + 64% + 24576%.

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

When the search is successful, the file MARKIE is available for input, by means of GET statements. Remember, since the file is open for input only, attempting to execute PUT statements results in a ?PROTECTION VIOLATION error (ERR = 10).

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

2.2 The File Structured Magnetic Tape OPEN FOR OUTPUT

The OPEN FOR OUTPUT statement searches for a specified file on a designated tape 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 tape 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. Note that account numbers are ignored on ANSI labeled tapes.

If the file is not found, the system writes a magnetic tape 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 tape. 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: **Magnetic Tape OPEN FOR OUTPUT MODE Values**

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

2.2.1 Searching for a Label on OUTPUT

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 file label record and its filename (and account for DOS tapes) matches that 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 file 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 file 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 magnetic tape. 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 destroy 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 magnetic tape.

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 tape at its current position. If the system finds a file label record and the filename in the label record matches the filename in the OPEN FOR OUTPUT statement, it backspaces over the file 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 file 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 magnetic tape open with MODE 16% writes trailing EOF (End-of-File) tape marks and 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 file label record at the current tape position. No label record reading occurs. The system simply writes a new file label record, destroying 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 file record label at the current tape position.

NOTE

> Any MODE value which includes 512% causes the 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 tape marks 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 filename specified in the OPEN statement does not match the label record, the system automatically rewinds the tape to the first file 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 file 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 file label record before reading any label. Once again, no match causes the system to check successive label records until the file is found or until the logical end-of-tape is detected. The system does not rewind the tape when it performs the CLOSE operation on channel N%.

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

2.2.5 DOS and ANSI Format Labels

MODE 16384% in the OPEN FOR OUTPUT statement causes the system to search for a magnetic tape 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 you omit the MODE value 16384%, the system assumes that the label records on the tape, either DOS or ANSI, are in the default format as specified by the system manager when time-sharing operations began or as specified by the user when the unit was reserved to the current job (ASSIGN or MOUNT). The MODE value overrides any current defaults for labeling.

MODE 24576% in the OPEN FOR OUTPUT statement causes the system to search for the magnetic tape file whose filename is specified. In this case, the file label must be written in ANSI format 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 file label by file label until it finds the correct file. If the logical end-of-tape is detected, the system automatically backspaces over the logical end-of-tape, 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 label records are in the system-wide or job-related default format.

2.2.6 Processing DOS Magnetic Tape 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 magnetic tape processing. PUT statements write blocks on tape equal to the buffer size – 512 bytes.

To write blocks larger than 512 bytes, 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 magnetic tape 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 Magnetic Tape Files

If the tape is being processed with ANSI labels, 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 has ANSI labels. The general form of the statement with options is:

```
10   OPEN 'MT0:ABC' FOR OUTPUT AS FILE N%,
     CLUSTERSIZE Q%, FILESIZE P%, 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 tape 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 in the example. The value 24576% in the MODE option ensures that ANSI label 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 file label 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 Magnetic Tape CLUSTERSIZE Values

Label Field Name	CLUSTERSIZE Value	Label Result
Record Format	0%	U (undefined*)
	16384%	F (fixed length)
	32767%+1%	D (variable length)
	-16384%	S (spanned) (**)
Record Length (in bytes)	Between 0% and 4095%	For U, always 0%. For F, value gives fixed record length. For D, value gives maximum record length. For S, value is unused. (**)
System Dependent (File Characteristics)	0%	M = carriage control embedded.
	4096%	A = FORTRAN carriage control.
	8192%	(space) = Implied carriage control (when printed, line feed precedes and carriage return follows each record).

* Undefined record format tapes cannot be interchanged with other operating systems.
** 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%. Note that U format records do not conform to ANSI standard X3.27–1978 and that non-RSTS/E operating systems cannot read tapes with undefined format.

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 18 (the minimum allowed on ANSI labeled tape) and 4095. Note that because a record cannot span blocks, the FILESIZE value for fixed length records must be a multiple of the CLUSTERSIZE value.

In ANSI label processing, the system uses the block length from the HDR2 label to create the magnetic tape I/O buffer. This action enables the program to write blocks of data 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 label 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 label processing appears similar to the RECORDSIZE option. However, if the RECORDSIZE option is used in ANSI label 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 label processing.

The PUT statement without the COUNT option in ANSI label 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 specifies in the FILESIZE option the desired block length (an even number between 18 and 512). The system writes the block length to the HDR2 label and establishes the I/O buffer equal to the block length. (This action differs from DOS label 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. (This action differs from DOS label processing where the COUNT or RECORDSIZE option is required to write blocks smaller than 512 bytes.) The system writes a block at the size of the I/O buffer (which matches the block length in the HDR2 label).

Data, which is to be written to ANSI labeled tape, is not automatically converted by RSTS/E to the appropriate ANSI record format. The application program must format the data in the I/O buffer before writing the buffer to the tape. Also, data read from an ANSI labeled tape must be interpreted in the appropriate ANSI record format by the program. It is not in the scope of this manual to fully describe ANSI record format; refer to the ANSI standard X3.27 - 1978. Note that the PIP utility can convert user data to the desired ANSI record format (see the *RSTS/E System User's Guide*).

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 magnetic tape file.

If RSTS/E encounters the physical 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 marker without error. RSTS/E provides two recovery procedures:

1. Close the file as soon as the error occurs, then create another file on another tape for the remainder of the data.
2. If one file is to contain all the data, include in the program a subroutine that writes a logical end-of-tape mark at the end of the previous file. You may then write the file that generated 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 the last end-of-file (EOF) or the 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 tape marks (see Section 2.8.2) to zero the tape, then try a longer tape. Encountering BOT should occur only on DOS tapes. ANSI tape files contain a tape mark between the label records; thus, the system should find a tape mark before encountering BOT.
   d. If the tape is at a tape mark and is in DOS format, write three tape marks. On an ANSI labeled tape, backspace to the next tape mark, then write three tape 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 tape unit 0 and associates it with channel 2%. MODE 16466% is the sum of MODE 2% + 16% + 64% + 16384%.

When line 10 is executed, the system determines whether the current label record 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 label record is written, the file LLL317 is available for output, by means of 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 Magnetic Tape OPEN

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

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

The system associates tape unit 0 with the internal channel designated by N% and searches for the file ABC 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 magnetic tape. OPEN FOR INPUT and OPEN FOR OUTPUT enable the program to determine immediately which operation is needed.

2.4 The File Structured Magnetic Tape CLOSE

The CLOSE statement terminates processing of a magnetic tape 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 Magnetic Tape 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 magnetic tape and the user program. Tapes of any format can be read or written with non-file structured magnetic tape 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. However, other operating systems may not be able to process records of less than 18 bytes, which is the minimum record length allowed by ANSI standard X3.27–1978. Attempting to write a shorter record results in the ?ILLEGAL BYTE COUNT FOR I/O error (ERR = 31). In the OPEN statement, only the tape unit is specified to indicate non-file structured processing; no filename is included. There are three types of OPEN statements:

```
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 tape 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 method of opening a tape for writing and the actions that occur on CLOSE are discussed in Section 2.6.

2.6 The Non-File Structured Magnetic Tape CLOSE

CLOSE has no special action on non-file structured tapes unless OPEN FOR OUTPUT was used. On a magnetic tape that was OPEN FOR OUTPUT, the CLOSE statement causes trailing tape marks to be written, followed by backspacing over two of these tape marks, to position the tape correctly for subsequent output operations.

In any case, if the tape 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 magnetic tape 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, TE10, or TS03 drive, MODE indicates parity only, since a 9-track TU10 or TE10 drive reads and writes only at 800 BPI. When used with a 9-track TU16, TE16, TU45, or TU77 drive, MODE also can indicate 1600 BPI phase encoded mode. Note that for the 9-track TS11, only 1600 BPI phase encoded mode is possible.

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

```
MODE E+(D*4)+P+S
```

where:

E (exended density) is:

256 =	see values of D where E equals 256.
0 =	see values of D where E equals 0.

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

if E equals 0, then the values of D are;

0 =	200 BPI (7-track only).
1 =	556 BPI (7-track only).
2 =	800 BPI (7-track only).
3 =	800 BPI, dump mode (7-track). 800 BPI (9-track).

if E equals 256, then the values of D are;

0 =	1600 BPI, phase encoded (9-track only).
1 =	reserved for future use.
2 =	reserved for future use.
3 =	reserved for future use.

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.*

Dump mode indicates that the system dumps from memory the actual representation of the data. Because 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.

* An optional feature patch may be installed to change system tape parity and density defaults. Consult the *RSTS/E V7.0 Release Notes* for details.

Other values for density are in non-dump mode and apply only to 7-track drives. On output operations, the system writes bits 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 tape.

Effectively, MODE for a 9-track TU10 or TE10 drive can only be a 0 or 1 because the system operates at 800 BPI dump mode with 9-track TU10 or TE10 drives.* If any other values are used, the system recognizes only the parity specification.

MODE for a 9-track TU16, TE16, TU45, or TU77 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 if the tape unit was assigned to 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%
```

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.

* An optional feature patch may be installed to change system tape parity and density defaults. Consult the *RSTS/E V7.0 Release Notes* for details.

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 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 the 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 for DOS and 18 bytes for ANSI (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 magnetic tape 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 tape 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 tape unit 1 has been opened on internal channel 2. 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%
```

Table 2–5 summarizes the MAGTAPE function codes and includes the designations IMMEDIATE and 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: MAGTAPE Function Summary

Action	Function Code	Parameter	Value Returned	Wait Immediate
Rewind and off-line	1	unused	0	Immediate
Write tape mark	2	unused	0	Wait
Rewind	3	unused	0	Immediate
Skip record	4	records to skip	# records not skipped	Wait
Backspace over record	5	records to backspace	# records not backspaced	Wait
Set density and parity	6	E+(D*4)+P+S	0	Immediate
Tape status function	7	unused	status	Immediate
File characteristics	8	unused	file characteristics	Immediate
Rewind on CLOSE	9	unused	0	Immediate

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

Function code	= 1
Parameter	= unused
Value returned	= 0

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

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

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

2.8.2 Write Tape Mark Function

Function code	= 2
Parameter	= unused
Value returned	= 0

The WRITE tape mark function writes one tape mark record at the current position of the magnetic tape. For example:

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

writes a tape mark on the magnetic tape that is open on internal channel 2.

2.8.3 Rewind Function

Function code	= 3
Parameter	= unused
Value returned	= 0

The Rewind function rewinds the selected magnetic tape. For example:

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

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

2.8.4 Skip Record Function

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

The Skip Record function advances 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 a tape mark is encountered (in which case the value returned is the specified number of records to skip minus the number actually skipped).* For example, to skip from the current tape position to just past the next tape mark, use the following function:

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

* The system counts the tape mark as a record skipped.

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

2.8.5 Backspace Function

Function code	= 5
Parameter	= number of records to backspace (1 to 32767)
Value returned	= number of records not backspaced (0 unless tape mark or BOT 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 tape marks 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).* The BOT is neither skipped nor counted as a skipped record. For example:

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

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

2.8.6 Set Density and Parity Function

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

where:

E (extended density) is:

256 = see values of D where E equals 256.
0 = see values of D where E equals 0.

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

if E equals 0, then the values of D are;

0 = 200 BPI (7-track only).
1 = 556 BPI (7-track only).
2 = 800 BPI (7-track only).
3 = 800 BPI, dump mode (7-track).
800 BPI (9-track).

if E equals 256, then the values of D are;

0 = 1600 BPI, phase encoded (9-track only).
1 = reserved for future use.
2 = reserved for future use.
3 = reserved for future use.

* The system counts the tape mark as a record actually backspaced.

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.

A tape 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 tape 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 tape 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. If this function is executed on a tape open in file structured mode, the request is ignored and the value returned is the same as the value passed.

By adding 8192% to the parameter value (making it 8192%+2%*4%+1%, in the 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 tape 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 tape 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 following immediate mode routine sets tape 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 tape 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

Function code	= 7
Parameter	= unused
Value returned	= status

The Tape Status function returns the status of the specified magnetic tape as a 16-bit integer, with certain bits set, depending on the current status. The format is shown in Table 2-6. The following example obtains the status of the magnetic tape opened on internal channel number 2:

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

Table 2-6: Magnetic Tape Status Word

Bit	Test	Meaning
15	I% < 0%	Last command caused an error.
14–13	(I% AND 24576%)/8192%	If bit 3=0, density: 0 = 200 BPI 1 = 556 BPI 2 = 800 BPI 3 = 800 BPI, dump mode If bit 3=1, density: 0=1600 BPI 1=reserved 2=reserved 3=reserved
12	(I% AND 4096%) = 0%	9-track tape. 640
	(I% AND 4096%) <> 0%	7-track tape. - 512
11	(I% AND 2048%) = 0%	Odd parity. 138
	(I% AND 2048%) <> 0%	Even parity. - 128
		10
10	(I% AND 1024%) <> 0%	Tape is physically write locked. - 8
9	(I% AND 512%) <> 0%	Tape is beyond end-of-tape marker. - 2
8	(I% AND 256%) <> 0%	Tape is at beginning-of-tape (Load Point).
7	(I% AND 128%) <> 0%	Last command detected a tape mark.
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% = 1%	Unit does not accept 1600 BPI. Unit accepts 1600 BPI.
3	(I% AND 8%) = 0% = 1%	See values for bits 14–13. See values for bits 14–13.
2-0	(I% AND 7%)	Indicates last command issued: 0 = OFF-LINE 1 = READ 2 = WRITE 3 = WRITE TAPE MARK 4 = REWIND 5 = SKIP RECORD 6 = BACKSPACE RECORD

When the value of I% returned is 17,409 decimal (or 42001 octal), the magnetic tape 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. For example:

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

   = 4 2 0 0 1 (octal)

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

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

```
    I% 100 010 000 000 001

AND 17409% 110 000 000 000 000

    Result 100 000 000 000 000
```

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

The results of (I% AND 4096%) and (I% AND 2048%) are both zero, indicating a 9–track tape with odd parity. (I% AND 1024%) results in a non-zero number in this case, so the tape 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 magnetic tape status function can advance to the next tape mark (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 tape mark 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

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

This function returns the status of the specified file structured magnetic tape 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 always returned for DOS files.

For example, to obtain the characteristics of a file on a magnetic tape 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 magnetic tape 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. For example:

```
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 tape file is in ANSI format F.

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

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

Table 2-7: Magnetic Tape File Characteristics Word for ANSI Format

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)**
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 " ")
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**

* U (undefined) format does not conform to ANSI standard X3.27-1978

** ANSI format S is not supported by RSTS/E systems.

2.8.9 Rewind On CLOSE Function

Function code	= 9
Parameter	= unused
Value returned	= 0

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

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

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

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

2.9 Magnetic Tape Programming Examples

2.9.1 Writing a Magnetic Tape File

The following BASIC-PLUS program opens an existing magnetic tape 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 Magnetic Tape File

The following program opens a magnetic tape 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 "MT0: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 magnetic tape 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 Magnetic Tape Non-File Structured

The following program reads a DOS magnetic tape 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 Magnetic Tape Error Handling

It is important to consider details of the system's handling of magnetic tape error conditions. These are: parity error, record length error, off-line (not ready) error, write lock error and write beyond EOT error. Other error conditions that can occur with magnetic tape (Illegal byte count, File exists, Protection violation) are described in Appendix C.

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 tape 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 tape status word is set. Therefore, if a program is reading records of unknown length from magnetic tape, 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 tape 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 magnetic tape.

2.10.3 Off-Line Error

The status of the tape unit is determined by testing bit 5 of the returned value of the tape status function shown in Table 2-2. If bit 5 is set, the tape 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 magnetic tape 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 magnetic tape files.

2.12 Magnetic Tape Special Function – SPEC%

The SPEC% function performs special operations on magnetic tape, flexible diskettes (see Section 1.5.5), disks (see Section 1.7), and terminals (see Section 4.6).

For magnetic tape, the SPEC% function allows you to rewind the tape, skip records on the tape, and set tape density and parity. Note that these operations are covered in detail in Section 2.8.

The SPEC% function for magnetic tape has the following format:

```
VALUE%=SPEC%(FUNCTION%,PARAMETER,CHANNEL%,HANDLER INDEX%)
```

where:

VALUE%	depends on the particular function code specified in FUNCTION%.
FUNCTION%	is the function code.
PARAMETER	depends on the particular function code specified in FUNCTION%.
CHANNEL%	is the I/O channel on which the operation is to be performed.
HANDLER INDEX%	is the handler index on the I/O device open on CHANNEL%. The index for magnetic tape is 14%.

The SPEC% function performs various operations on magnetic tape as determined by the code you specify in FUNCTION%. These operations duplicate those performed by the MAGTAPE function codes and are described in Table 2-5. That is, the format of the MAGTAPE function:

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

parallels the following SPEC% format:

```
I%=SPEC%(FUNCTION%-1%,PARAMETER,CHANNEL%,14%)
```

Chapter 3
Line Printer

3.1 Special Character Handling

Certain non-printing 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. Note that the system manager must set the LP CONTROL characteristic during system initialization to enable the use of LS11 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.)

(continued on next page)

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

Character	LS11 Resultant Action	LP11 Resultant Action
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$(12)	FF - Form Feed 1. Prints line 2. Returns carriage 3. Advances paper to the next hole position in Channel 7 (see Section 3.2)	FF - Form Feed 1. Prints line 2. Returns carriage 3. Advances paper to the third line of the next form (hardware top of form, see Section 3.2)
CHR$(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 you 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.

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 (zero) to the character O ("oh").
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 six lines (i.e., over perforation line) at the bottom of each form.
4096%	Moves paper to top of hardware form. CHR$(12%). (See discussion, in Section 3.2.2.)
8192%	Suppress form feed on CLOSE.

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

3.2.1 Handling Non-Standard Forms - MODE 512% + N%

The MODE value 512% allows a program to control non-standard length forms in the line printer.* 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.

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.

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

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, 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 as shown in Appendix D. 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%).

3.2.2 Hardware Form Feed - MODE 4096%

A line feed (LF) character sent to the printer advances the form to the first print position of the next line (i.e., LF implies carriage return (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$(96) and CHR$(122) to their equivalents between CHR$(65) and CHR$(90). 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 2048%+1024%+512%+256%+128%+66%
```

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 you load the line printer 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 'LP0:' 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 as follows:

```
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 6 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 of 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 carriage return (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 to top of form* and to the left margin respectively.

3.4 Operation and Error Handling

An error condition at the line printer causes the system to interrupt the transfer of data from the buffers to the device, but not 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 appropriate error handling. The system indicates a 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 because of the error condition. Because all of the data is buffered, you should not write exceptionally large buffers to the line printer. Every 10 seconds the Monitor checks the printer's status. It will resume printing when the error condition is removed. To prevent filling up monitor buffer space, subsequent output requests return immediately with ERR=14. No data is buffered while the error condition persists. When an output request returns without error, the printer error is cleared. However, good programming practice is to force the Monitor to delay until line printer output is complete before printing any more data.

The following sample program 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.

* The driver'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.

```
10     !   HOUSEKEEPING
20     OPEN "DATA.DAT" FOR INPUT AS FILE 1%
       \OPEN "LP0:" AS FILE 2%, RECORDSIZE BUFSIZ(1%)
       \FIELD 1%, BUFSIZ(1%) AS I$
       \FIELD 2%, BUFSIZ(2%) AS O$
       \FIELD 2%, 1%, AS O1$
       \E% = 0%
       \ON ERROR GOTO 200
100    !   COPY LOOP
110    GET #1%
       \C% = RECOUNT
       \LSET O$ = I$
120    PUT #2%, COUNT C%
       \GOTO 100
130    !LINE PRINTER OUTPUT ERROR - DATA PUT
       !AT LINE 120 IS BUFFERED
140    LSET O1$ = CHR$ (0%)
150    PUT 2%, RECORD 4%, COUNT 1%
       \E% = 0%
       \PRINT IF POS (0%)
       \GOTO 100
       !   PUT A NULL (IGNORED BY MONITOR)
       !   AND WAIT FOR PRINTER READY
       !   IF IT MAKES, PRINTER IS OK, SO GO
       !   BACK TO COPY LOOP
160    PRINT 'PRINTER HUNG - PLEASE FIX IT';
               UNLESS E%
       \PRINT CHR$ (7%);
       \E% = -1%
       \SLEEP 10%
       \GOTO 150
       !   ASK FOR REPAIRS ONCE, DING EACH
       !   TIME, SLEEP AND RETRY
200    !   ERROR HANDLING
210    RESUME 300   IF ERR = 11%  AND ERL = 110%
       \RESUME 130  IF ERR = 14%  AND ERL = 120%
       \RESUME 160 IF ERR = 14%  AND ERL = 150%
       \ON ERROR GOTO 0
300    !   DONE
310    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 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 the expiration of a Receive call's wait time (see Section 8.5). The following sample routine shows the procedure for cancellation on terminal input.

```
100   OPEN "KB:" AS FILE #1%
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   SLEEP 5%\GOTO 110
```

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. You can establish the terminals as slave terminals with the ASSIGN command before you run the program. Alternatively, the program can assign these terminals by executing the Assign/Reassign device SYS system function call to FIP (see Section 7.2.4.3). 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, as described in the following sections. 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 reissuing the GET statement continues operation. A race condition can occur when two jobs are accessing the same data. That is, one job attempts to access data while another job is in the act of changing that data. The system is unable to resolve these two simultaneous conditions.
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 occurs.
S%=8192%	If no input is pending from any of the terminals, the ?USER DATA ERROR ON DEVICE error 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.

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 TECO.
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%).
16%	Guards program against CTRL/C interruption and dial-up line hibernation.
32%	Enables incoming XON/XOFF processing.
64%	Reserved for future implementation.
128%	Special scope RUBOUT.

The following sections describe the various MODE options.

4.3.1 Binary Data Output and Input – MODE 1%

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%
```

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% in the PUT statement, all output formatting on the terminal is disabled.

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

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

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

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

```
GET #N%
```

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

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

Since the system accepts and does not alter any characters received from a terminal open for binary input, typing CTRL/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 by means of 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%*

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 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 non-zero 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.)

* Echo control is an optional feature of the RSTS/E Monitor and may not be available on all systems.

MODE 8% also turns off other MODE options in effect (except MODE 16% and MODE 128%) 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). 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.

(continued on next page)

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

Class	ASCII Code	Code Returned to User	Comments
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.
Synchronization	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 (CTRL/U or DELETE) 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% is in the range of 1% to the size of the buffer declared on channel C% and indicates how many bytes in that buffer represent the field declaration. The field is defined as follows:

a. If N% = 1% The byte contains the field size and overflow handling information. The field size must be in the range of 1 to 127. If you attempt to declare a size of 0, the system returns the ?ILLEGAL BYTE COUNT FOR I/O error (ERR = 31).

If you add 128 to the field size, it indicates that keypunch overflow handling is to be used instead of normal overflow handling.

b. If N% = 2% The first byte contains the field size declaration as described in N% = 1%.

The second byte contains the ASCII value of the paint character. If this byte is 0, or N% = 1%, then a space is the paint character by default.

c. If N% > 2% The first N minus 2 bytes contain a prompt that is to print on the terminal before the field.

Byte N minus 1 is the field size declaration as described in item a. The last byte is the paint character as described in item b.

For example:

COUNT 1% specifies that the first byte in the buffer declares the field size. Space becomes the paint character by default.

COUNT 2% specifies that the first byte in the buffer declares the field size. The second byte in the buffer declares the paint character. If a space is desired as a paint character, specify 0% or the ASCII value for space in this byte.

COUNT 20%	specifies that the first 18 bytes in the buffer contain the prompt. The prompt is a string of ASCII characters. Byte 19 in the buffer contains the field size. Byte 20 in the buffer contains the paint character.

The PRINT statement can also be used to declare a field with a method similar to that of the PUT statement. The PRINT statement includes a RECORD 256% modifier to indicate the field declaration and string specifications (in place of the COUNT option) to declare field parameters. Consider the following:

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

where:

C%	is the non-zero channel open with MODE 8%.
M%	is the overflow handling code. M% = 128% for keypunch. M% = 0% for normal.
S%	is the field size in the range of 1 to 127.
+	concatenates the field declarations.
'P'	is the ASCII paint character.
A$	is the prompt.
P%	is the decimal code for the paint character (for example, underline is CHR$(95%)).
;	terminates the string (suppresses CR/LF).

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 (or the nth character for an n-character keypunch field) is received, the field is deactivated. Echoing is 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). 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 sequence is recommended when interacting with a scope terminal in echo control mode.

1. Open any terminal on a non-zero channel with MODE 8%.
2. Execute SYS call 11 to cancel type ahead (see Section 7.1.2.11). 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, prompt, and 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 non-zero channel with MODE 8%.
2. Execute SYS call 11 to cancel type ahead see Section 7.1.2.11.
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, overflow handling mode, and prompt. 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.

4.3.4 Prevent CTRL/C Interruption and Hibernation - MODE 16%

When an OPEN statement includes MODE 16%, the program is protected from aborting when CTRL/C is typed at the terminal and from detaching and hibernating when it is running on a dial-up line that gets hung up.

Typing CTRL/C at a terminal that is open with MODE 16% cancels any pending output to the terminal, sets CTRL/O, and is interpreted as an ASCII 3. The program can recover and continue output.

Hanging up a dial-up line (without using MODE 16%) would cause the job to be detached and to enter the hibernation state as soon as it did terminal I/O. That is, the job would wait until it is attached, through some external process, before it could recover. With MODE 16%, an immediate exit to ERR=27 (?I/O TO DETACHED KEYBOARD) occurs when terminal I/O is attempted, to allow the program to initiate immediate recovery.

MODE 16% 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 16%, or
3. any I/O is performed on channel 0 (the job's console terminal).

4.3.5 Enable Incoming XON/XOFF Processing - MODE 32%

When an OPEN statement includes MODE 32%, an incoming XOFF character (ASCII 19) suspends ouput to the terminal; an incoming XON character (ASCII 17) resumes output to the terminal.

When the OPEN statement also includes MODE 1% (for binary input), all other incoming characters are processed as for MODE 1%. However, all other incoming characters are ignored when the OPEN statement does not also include MODE 1%.

MODE 32% processing remains in effect until one of the following conditions are met:

1. a CLOSE is performed on the channel,
2. the terminal is opened again without MODE 32%,
3. and I/O is performed on channel 0 (the job's console), or
4. an input timeout occurs, producing ERR=15.

4.3.6 Special Use of RUBOUT - MODE 128%

When an OPEN statement includes MODE 128%, it allows CRT (scope) terminals to use RUBOUT as a delimiter. RUBOUT's use as a delimiter is subject to the following conditions:

1. If a typed character is the object of a RUBOUT operation and that character is a graphic in the range of CHR$(32) to CHR$(126), a normal scope character deletion is performed.
2. If there is no typed character or if the character is a non-graphic in the range of CHR$(0) to CHR$(31) and CHR$(127), no character is deleted. The RUBOUT is buffered as a delimiter and the job is scheduled.

The ability of MODE 128% to buffer RUBOUT as a delimiter is particularly useful to TECO and other scope-oriented editors.

CAUTION

MODE 128% is reserved for use with DIGITAL-supplied software and it is subject to change in future releases.

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 ($) character. The system does not translate the CHR$(155) character or 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$(127)) are processed first. Although embedded in an escape sequence, their functions remain unchanged. Note that control characters in escape sequences are a violation of the ANSI standard and should not be used. The control character CHR$(27) itself triggers the start of a new escape sequence. The function of control characters DELETE CHR$(127) and CTRL/U CHR$(21) do change. They are discarded and not passed to the user.
2. Within an escape sequence, any normal data conversion, such as translating lower case to upper case, is not done.

3. If an escape sequence has been triggered but not terminated, the system continues to process it.
4. 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 <Y> <y-addr> <x-addr>
   ```

   (This form is used by some terminals for direct cursor addressing.)

2. O (CHR$(79)), P (CHR$(80)), or ? (CHR$(63)) followed by one trailing character

   ```
   ESC <O> <modifier>
   ESC <P> <modifier>
   ESC <?> <modifier>
   ```

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

3. [, right bracket (CHR$(91)), followed by any number of filler characters (in the range CHR$(32) through CHR$(63)), and a terminating character in the range of CHR$(64) through CHR$(126)

   ```
   ESC<[><n fillers CHR$(32)-CHR$(63)><CHR$(64)-CHR$(126)>
   ```

4. The triggering escape character, followed by any number of filler characters (in range CHR$(32) through CHR$(47)), and a terminating character in the range of CHR$(48) through CHR$(126)

   ```
   ESC <n fillers CHR$(32)-CHR$(47)><CHR$(48)-CHR$(126)>
   ```

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 </> <Z>
```

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

```
<CHR$(128)> </> <Z> <CHR$(155)>
```

The ESC character is translated to CHR$(128). The slant character is a filler and is passed to the program. The Z character, which is not a control or a filler character, terminates the escape sequence. The system passes the CHR$(155) character to the program to signal the termination of the escape sequence.

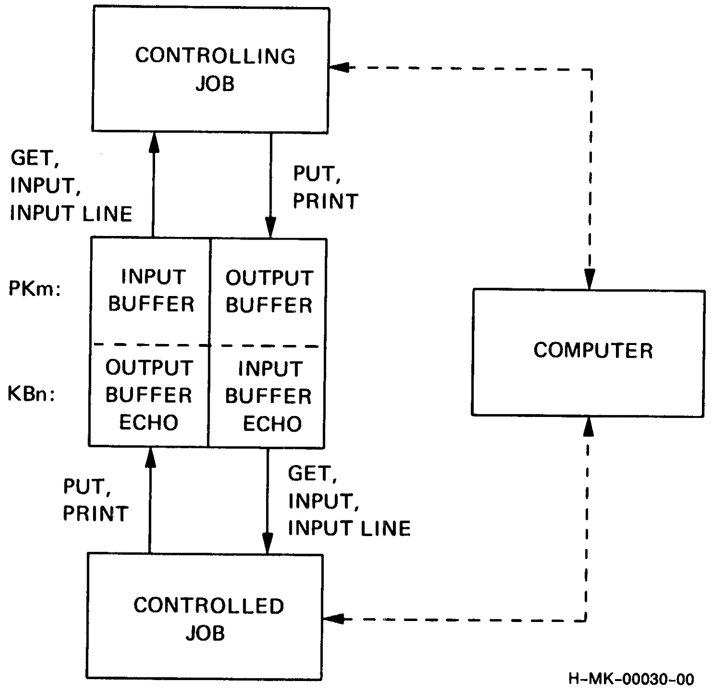
4.5 Pseudo Keyboard Operations

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:, where 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.

Figure 4-1: Pseudo Keyboard Operations



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 carriage return character (CHR$(13)) sent to the PK input buffer returns from the KB buffer as a carriage return and line feed sequence.

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.

Note that if the pseudo keyboard OPEN statement specifies MODE 0%, the controlled job is killed when the keyboard is closed. Also, if the pseudo keyboard OPEN statement specifies MODE 1%, the controlled job is detached when the keyboard is closed. For example:

```
100  OPEN "PK3:" AS FILE 1%, MODE 0%
200  OPEN "PK5:" AS FILE 2%, MODE 1%
300  CLOSE 1%, 2%
```

When these statements execute, the job that is running on PK3: is killed and the job on PK5: is detached. Note that the MODE 1% option can be made a privileged function.

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 the 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 error (ERR = 11) 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%
```

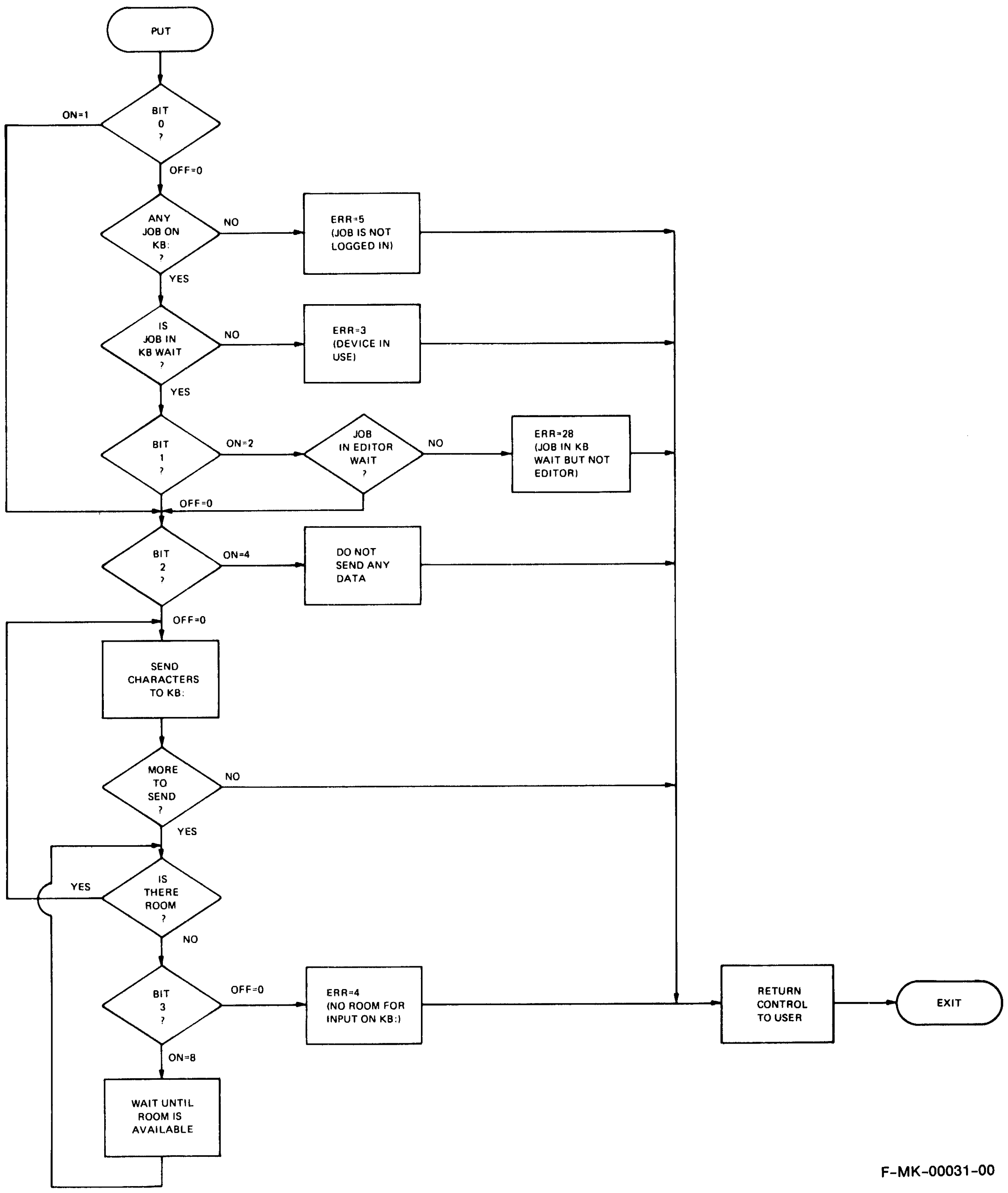
The value N% is the number of the internal channel on which the PK device is open. The value of C% is the number of bytes the program sends from the buffer to the pseudo keyboard. Without the COUNT option, the PUT statement transmits either the number of bytes specified in the RECORDSIZE option of the 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 carriage return character (CHR$(13%)) as a terminator and the value C% must reflect the carriage return character in the total number of bytes sent. The system automatically appends a line feed character to a line terminated by a carriage return 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.

Figure 4-2: PUT Statement Actions for Pseudo Keyboard Output



F-MK-00031-00

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 at command level. If it is waiting for KB input but is not at command level, the system generates ERR=28 (?PROGRAMMABLE ^C TRAP). 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.

The RECORD 16% option in a PUT statement is used to kill any job that is currently running on a specified pseudo keyboard. That is, the controlling job specifies the controlled job's keyboard in the PUT statement and specifies RECORD 16% to kill that job. For example:

```
100   PUT #8%, RECORD 16%
```

kills the job that is currently running on the PK: unit open on channel 8.

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 an ?ACCOUNT OR DEVICE IN USE error (ERR=3).

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.

4.6 Terminal Special Function – SPEC%

The SPEC% function performs special operations on terminals, flexible diskettes (see Section 1.5.5), disks (see Section 1.7), and magnetic tape (see Section 2.12).

For terminals, the SPEC% function allows you to cancel CTRL/O and set modes for tape, echo, and ODT.

The SPEC% function for terminals has the following format:

```
VALUE%=SPEC%(FUNCTION%,PARAMETER,CHANNEL%,HANDLER INDEX%)
```

where

VALUE%	depends on the particular function code specified in FUNCTION%.
FUNCTION%	is the function code. The SPEC% function performs various operations on terminals as determined by the FUNCTION% code. These codes are as follows:

FUNCTION%=0	cancel CTRL/O.
FUNCTION%=1	set tape mode.
FUNCTION%=2	enable echo and clear tape mode.
FUNCTION%=3	disable echo.
FUNCTION%=4	set ODT mode.
FUNCTION%=7	cancel all type ahead.

PARAMETER	specifies the terminal on which the operation is to take place. If PARAMETER is 0, the operation is performed on the currently open terminal. If you specify a keyboard number in PARAMETER, the operation is performed on that terminal. Note that the keyboard must be assigned to the calling job but must not be open.
CHANNEL%	specifies the I/O channel for the terminal in PARAMETER.
HANDLER INDEX%	the handler index of the I/O device open on CHANNEL%. The index for terminals is 2%.

Chapter 5
DECtape

5.1 File Structured DECtape — TU56

The user program indicates file structured processing on TU56 DECtape by including a filename with the device specification in the OPEN statement. Up to 12 files can be open for read access simultaneously on a TU56 DECtape drive. Only one file, however, can be open for write access on a TU56 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 TU56 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 TU56 DECtape block because the system treats the remaining 2 bytes as a pointer to the next block in the file. (Refer to Figure 5-1.) 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 DECtape — TU56

In non-file structured processing of TU56 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 the first 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 because it attempts to create a file.

After opening a TU56 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%
```

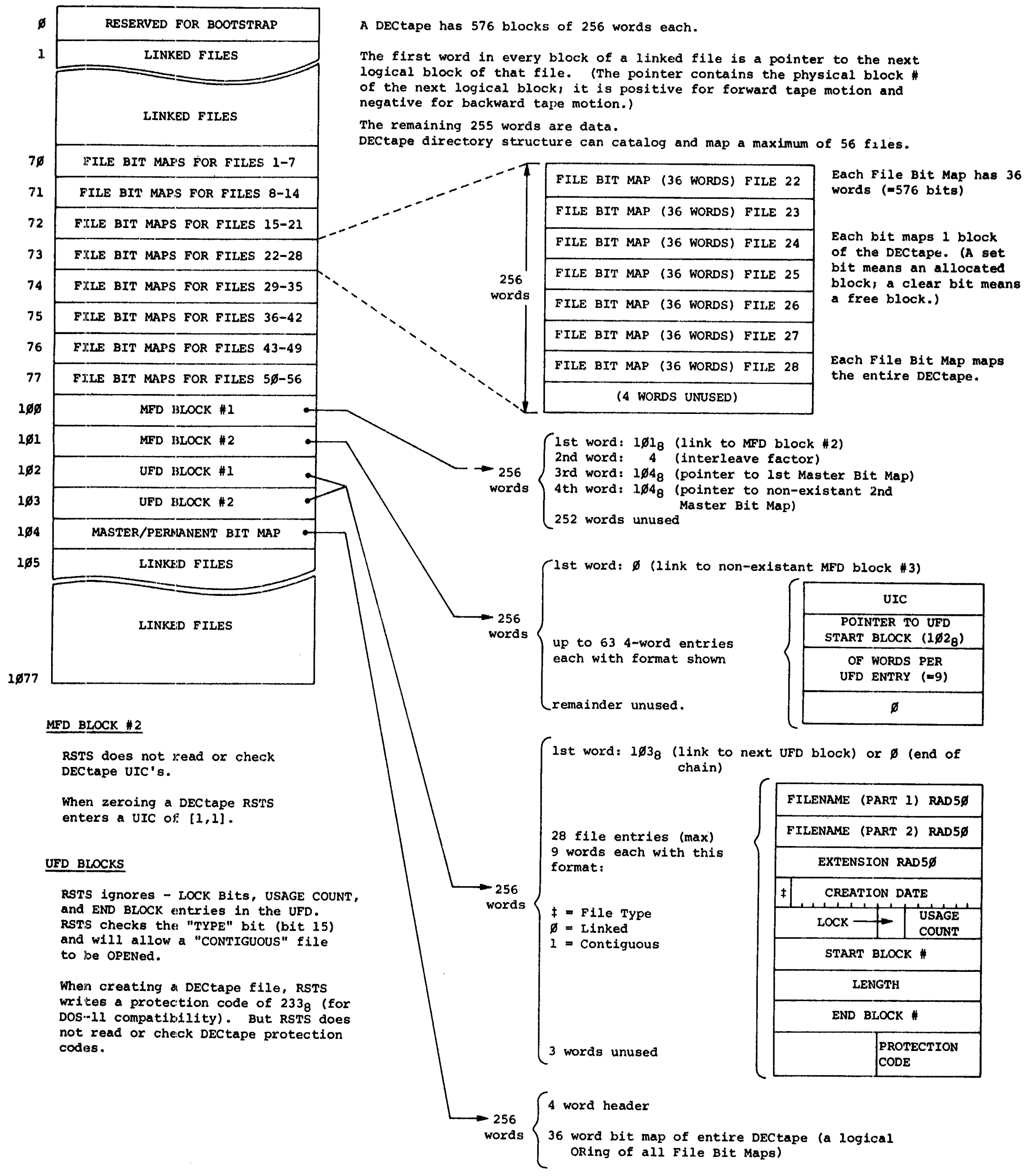
reads block 4 of the file opened on channel 1% backwards. The maximum block number is 578.

In writing non-file structured TU56 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 TU56 DECtape.

Because the blocks are not physically contiguous in file structured mode, 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 TU56 DECtape format diagram (Figure 5-1) 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.

Figure 5-1: TU56 DECtape Format



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
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
256 words
1st word: Ø (link to non-existant MFD block #3)
up to 63 4-word entries each with format shown
remainder unused.
UIC
POINTER TO UFD START BLOCK (1Ø2$_8$)
OF WORDS PER UFD ENTRY (=9)
Ø
256 words
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
FILENAME (PART 1) RAD5Ø
FILENAME (PART 2) RAD5Ø
EXTENSION RAD5Ø
‡ CREATION DATE
LOCK
USAGE COUNT
START BLOCK #
LENGTH
END BLOCK #
PROTECTION CODE
256 words
4 word header
36 word bit map of entire DECtape (a logical ORing of all File Bit Maps)

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.

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.

F-MK-00032-00

5.3 Non-File Structured DECtape II — TU58

The TU58 DECtape II is a block structured, mass storage, random access device. The device contains 512 blocks (each of which is 512 bytes long) and, as with the TU56 DECtape, specific blocks on the tape can be accessed. However, unlike the TU56, only non-file structured processing is allowed on the TU58.

To initiate processing on the TU58, the following two forms of the OPEN statement are allowed:

```
200  OPEN "DD1:" AS FILE 4%
```

and

```
200  OPEN "DD1:" FOR INPUT AS FILE 4%
```

The OPEN FOR OUTPUT statement is not allowed.

After opening the TU58 DECtape II, BASIC-PLUS GET and PUT statements can be used to read and write specific physical blocks on the tape. For example:

```
300  GET 4%, RECORD 6%, COUNT N%
```

and

```
500  PUT #4%, RECORD R%, COUNT N%
```

where COUNT N% is greater than zero. The value for RECORD is interpreted as the block number and is in the range of 0 to 511. Note that if the COUNT value in a PUT statement is not a multiple of 512 bytes, the remainder of the last block written is padded with NULs. Also, unlike the TU56, TU58 DECtape II does not allow the program to read or write backwards (the RECORD value must be positive).

Chapter 6
Card Reader and Paper Tape

6.1 Card Reader

The card reader reads data from standard (80-column) punched cards. Data is read from the card one column at a time in one of three modes: ASCII (Section 6.1.1), packed Hollerith (Section 6.1.2), or binary (Section 6.1.3). One card can be read (and the data on it stored) in any mode.

6.1.1 ASCII Mode

The card reader reads cards punched with the standard ASCII codes, as shown in Appendix B. One of four sets of codes may be used: ANSI, 029, 026, or 1401. 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 the *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
<
```

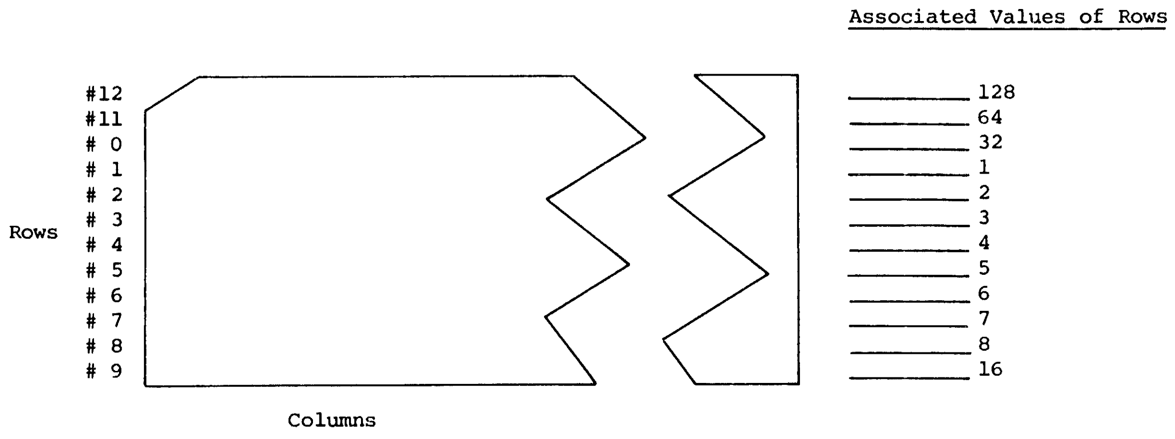
In this example the trailing spaces in card columns 27 through 80 are deleted, and the two characters, carriage return and line feed are added, making a total of 28 characters in the string A$.

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 (ERR=13) occurs. With INPUT or INPUT LINE statements, any columns containing illegal punches are stored as BACKSLASH (ASCII 92) 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 ASCII 220 code (BACKSLASH with the parity bit set). By checking the characters for code 220, the program can determine in which column(s) the error(s) occurred.

6.1.2 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.

Figure 6-1: Packed Hollerith Read Mode



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

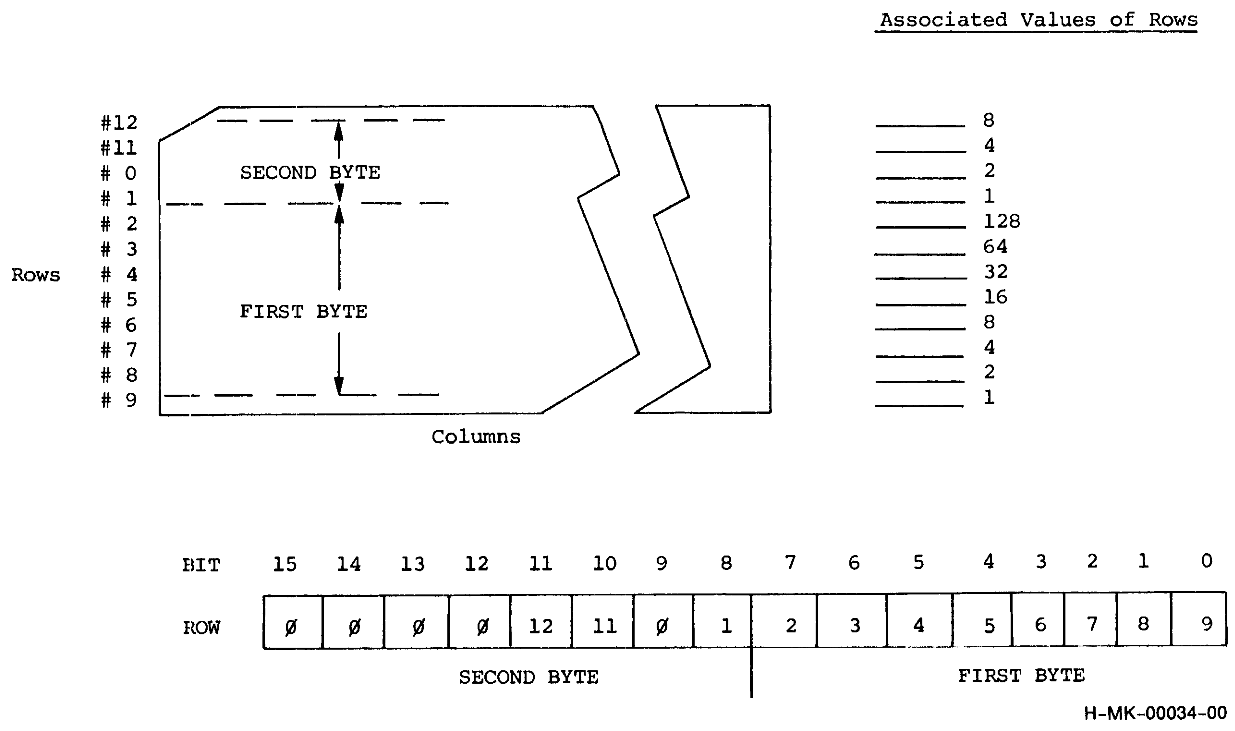
H-MK-00033-00

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.

6.1.3 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.

Figure 6–2: Binary Read Mode



6.1.4 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 in Table 6-1. 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.

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

For example:

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

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

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

```
                                                      Specified Read Mode
                                                         at This Point

100     OPEN "CR:" FOR INPUT AS FILE 1%, MODE 1%         Hollerith
        \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
        \GET #6%                                         ASCII
        \GET #6%, RECORD 258%                            Binary
        \CLOSE 6%
32767   END
```

Line 100 of the example program sets the read mode to Hollerith, then overrides it, setting the read mode to ASCII temporarily. When the last statement on the 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. However, a RECORD 258% option changes the mode to binary. Closing a file 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.

6.2 Paper Tape

The paper tape reader and punch read and punch data onto unoiled paper tape. In a punch operation, characters are read from the user's buffer and translated into perforations on the tape. In a read operation, the tape perforations are translated back into characters and written to the user's buffer. You can specify two modes to set even or odd parity for the characters punched on the tape (Section 6.2.1) and to check the parity of the characters read from the tape (Section 6.2.2).

6.2.1 Punching with Parity on Paper Tape

You can specify a mode value in an OPEN statement (with the MODE option), which allows you to control the parity of data punched on the paper tape.

Bit 1 of the mode word (value 2) enables the punching of paper tape with a generated parity. If this bit is not set, the paper tape punch (PP:) driver passes 8-bit characters from the buffer to the tape without a parity bit. If bit 1 is set, the PP: driver generates the parity bit for each character from the buffer and passes the parity and character to the tape.

Bit 0 of the mode word (value 1) specifies the parity that is to be passed with each character. If bit 0 is not set, characters are passed to the tape with even parity. If bit 0 is set, characters are passed to the tape with odd parity. As each character is punched, the high order bit is removed and the appropriate parity bit is added to the character.

Note that bit 0 is ignored if bit 1 is not set.

Consider the following:

```
MODE 0 - No parity punched.
MODE 2 - Even parity.
MODE 3 - Odd parity.
```

6.2.2 Parity Checking on Paper Tape

You can specify a mode value in an OPEN statement (with the MODE option), which allows you to check the parity of paper tape data.

Bit 1 of the mode word (value 2) enables parity checking. If this bit is not set, the paper tape reader (PR:) passes 8-bit characters from the tape to the user's buffer without checking parity. If bit 1 is set, the PR: driver performs parity checking on the characters that it reads.

Bit 0 of the mode word (value 1) specifies the expected parity of the data on the tape. If bit 0 is not set, the driver checks the tape for even parity. If bit 0 is set, the driver checks for odd parity. As the PR: driver reads the tape, it checks the parity of each character. If the parity is as specified, the driver removes the high order bit (the parity bit) and writes a 7-bit character to the user's buffer. If the parity does not match the specification, the driver writes a 7-bit character with the high order bit set. After the tape is read, any characters in the buffer with bad parity return a ?USER DATA ERROR ON DEVICE error (ERR=13).

Note that bit 0 is ignored if bit 1 is not set.

Consider the following:

MODE 0 - no parity check.
MODE 2 - check for even parity.
MODE 3 - check for odd parity.

Note that the RECOUNT variable (see the *BASIC-PLUS Language Manual*) contains the number of characters read after every input operation. With this information, your program can scan the buffer for any characters with a bad parity flag.

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–PLUS language. As such, the calls are system dependent and have calling formats different from any BASIC PLUS language call. Second, the SYS format allows the usage of a variable number of parameters.

SYS calls are separated into two classes: privileged and non-privileged. The privileged calls can be used only by a privileged user or by a privileged program. The non-privileged calls can be used by anyone and are completely safe in the sense that their misuse can do no damage to other programs 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 fourteen, 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 CTRL/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.
12	Return information on last opened file.
13	Reserved for special implementations.
14	Execute CCL command.

7.1.2 General SYS System Function Calls

(F=0)

7.1.2.1 Cancel CTRL/O Effect on Terminal — Not Pr'vileged

Data Passed:

Byte(s)	Meaning
1	CHR$(0%), the cancel CTRL/O effect code.
2	CHR$(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 affected 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 specified terminal. The terminal is selected by the 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 non-zero 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 affected 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 the 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 non-zero 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$(2%), the enable eohoing code.
2	CHR$(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 affected 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 the 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 non-zero 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 affected 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 the 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 non-zero 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 affected 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 carriage 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.

Because this function is used in the system program ODT.BAS and the debugging routine ODT.OBJ, it is sometimes referred to as "ODT submode". The terminal is selected by the 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 non-zero 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, Section 7.1.2.9.

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.

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$(9%), the exit and set up NONAME code.
2–3	The first 3 characters of the run-time system name, in Radix–50 format, to which control is to pass. If bytes 2–5 are zero, your private default run-time system is assumed.
4–5	The last 3 characters of the run-time system name, in Radix–50 format, to which control is to pass.
6	If not specified, the named run-time system is established as the job's private default run-time system. Otherwise, CHR$(N%), where N% is: 255% establish the run-time system as the job's private default run-time system. 0% enter the specified run-time system without establishing it as the job's default run-time system.

Data Returned: No data is returned.

Discussion: This call clears the current program from memory and returns control to the user's private default run-time system or the run-time system specified in bytes 2–5. 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 BASIC–PLUS 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. If byte 6 is specified with a value of 0, it causes a temporary switch to the run-time system named in bytes 2–5.

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 affected 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 non-zero channel number (on which the master terminal is open) and byte 3 denotes the keyboard number of the slave terminal.

7.1.2.12 Return Information on Last Opened File — Not Privileged

Data Passed:

Byte	Meaning
1	CHR$(12%), the return information on last opened file code.

Data Returned:

Byte(s)	Meaning
1	The current job number times 2.
2–3	Internal coding.
4	The most significant bits of the file size (MSB size). If a non-zero number is returned, it indicates a large file. That is, the file size is greater than 65535 blocks.
5–6+	Project-programmer number.
7–10+	Filename in Radix–50 format.
11–12+	Filename extension in Radix–50 format.
13–14+	The least significant bits (LSB) of the file size (in blocks).
15–16+	The default buffer size (in bytes).
17–18+	The OPEN MODE value.
19–20	"Status".
21+	File cluster size (MOD 256).
22+	Protection code of the file opened.
23–24+	The physical device name, in ASCII format.
25+	The device's unit number (a real number).
26	Bit flags that specify whether or not the device is part of the public structure. See discussion.
27–30	Internal coding.

Discussion: When a compiled program is executed under the BASIC–PLUS Run-Time System, the full name of the program and the account in which it is stored is preserved by BASIC–PLUS at the time the file is opened. The information is retained until another file is opened, at which time the last-file-opened information is updated.

The program name and account information is preserved in filename string scan format and also includes information on the last .BAS or .BAC file which has been RUN, chained to, or entered as a CCL command line.

Byte 26 of the returned string contains the following information in bits 1 and 0 (the other bits are meaningless).

Bit 0=0	The device is in the public structure.
Bit 0=1	The device is a private disk.
Bit 1=0	A specific device was not specified.
Bit 1=1	A specific device was specified.

Examples:

1. DB3: is a public disk. If the file SY:FOO was last opened and the file is on DB3:, bytes 23–25 contain DB3. However, the program could examine byte 26 (using the AND operator) to determine that:

   Byte 26 AND 1 = 0 (the device is part of the public structure)

   Byte 26 AND 2 = 0 (the public structure was specified)

   Therefore the correct device designator is SY:.

2. DB3: is the public disk. Using DB3:FOO as last opened file, the correct device designator would be DB3: since:

   Byte 26 AND 1 = 0 (the device is part of the public structure)

   Byte 26 AND 2 = 2 (the device has a specific unit number – in byte 29)

NOTE

This call returns information about the file last opened, no matter how it was opened. For example, if the call is made after you have typed:

```
OLD PROG
RUN
```

the last file opened is the compiler work file, not the program PROG.BAS. Also, a program interrupted by CTRL/C and then reinitiated returns the last file opened before the CTRL/C when the call is issued.

7.1.2.13 Execute CCL Command — Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(14%), the execute CCL command code.
2–128	The string to be executed.

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

Possible Errors:

Meaning	ERR Value
?LINE TO LONG The string that was passed is too long to be executed as a CCL command. Note that the Monitor expands CCL abbreviations to their full syntax.	47
?ILLEGAL NUMBER A non-numberic value was used as an argument in one of the CCL switches. For example, a /SIZE:A switch specification can cause this error.	52
?ILLEGAL SWITCH USAGE An illegal switch was specified for the CCL command. For example, requesting a size that is larger than the system's SWAP MAX can cause this error.	67

Discussion: This call causes the Monitor to scan the string in bytes 2–128 to determine if it is a valid CCL command. If the string is valid, the command is executed as though it had been typed directly to a keyboard monitor. If the string is not valid because of one of the previously described error conditions, the program terminates (unless an error handling routine is in effect). If the string is valid but no such CCL command is defined, the monitor returns control to the caller (with no error) at the next program statement. Note that an optional patch is available that allows the system manager to disable this call; refer to the *RSTS/E V7.0 Release Notes.*

Additional errors can be detected after the current program is removed and execution of the CCL is attempted (see the *RSTS/E System Directives Manual*).

This page intentionally left blank.

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%) + "DKO: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 their previous use.

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 shown in Section 7.2.1, 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. The user program can transform the returned string to a 30-byte list using a CHANGE statement:

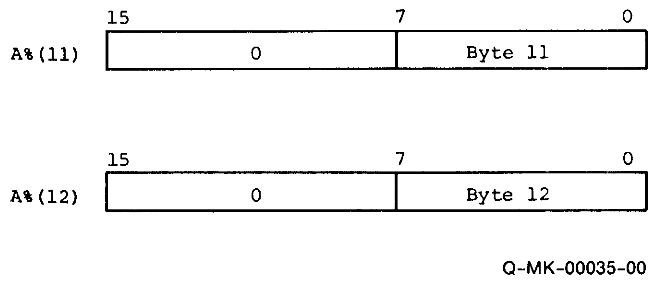
```
CHANGE A$ TO A%
```

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

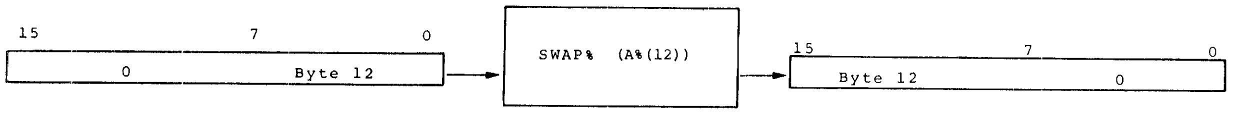


A%(11) contains the low byte portion of the Radix-50 word; A%(12) contains the high byte portion of the Radix-50 word. The two bytes must be combined into a single word and converted to the proper character string representation. This is accomplished by the following:

```
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 filename extension. Refer to the *BASIC-PLUS Language Manual* 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 the *BASIC-PLUS Language Manual* 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 following statement:

```
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 Filename 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, look up 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 look up call on the file DK0:[10,20]INVENT.DAT using the filename string scan FIP call.

```
10   DIM A%(30%)
     \A$="DK0:[10,20]INVENT.DAT"
     \CHANGE SYS(CHR$(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.

Table 7-2: **FIP SYS Calls (by Sub-Function Code)**

Function Code(F0)	Privileged* Status	Function Name	Page
-28	No	Spooling.	7-143
-27	Yes	Snap Shot Dump.	7-146
-26	Yes	File Utility Functions.	7-136
-25	No	Read or Write attributes.	7-123
-24	Yes	Add/Delete CCL command.	7-59
-23	No	Terminating filename string scan.	7-24
-22	Yes	Set special run priority.	7-63
-21	Yes	Drop/Regain (temporary) privileges.	7-65
-20	Yes	Lock/Unlock job in memory.	7-64
-19	Yes	Set number of logins.	7-88
-18	Yes	Add run-time system.	7-111
	Yes	Remove run-time system.	7-113
	Yes	Load run-time system.	7-114
	Yes	Unload run-time system.	7-116
	Yes	Add resident library.	7-117
	Yes	Remove resident library.	7-119
	Yes	Load resident library.	7-120
	Yes	Unload resident library.	7-122
-17	No	Name run-time system.	7-110
-16	Yes	System shutdown.	7-45
-15	Yes	Accounting dump.	7-92
-14	Yes	Change system date/time.	7-46
-13	Yes	Change priority/run burst/job size.	7-61
-12	No	Get Monitor tables - Part II.	7-105
-11	Yes	Change file backup statistics.	7-75
-10	No	Filename string scan.	7-24
-9	Yes	Hang up a dataset.	7-47
-8	No	FCB/DDB Information.	7-106
-7	No	CTRL/C Trap enable.	7-42
-6	Yes**	Poke memory.	7-87
-5	Yes	Broadcast to terminal.	7-48
-4	Yes	Force input to terminal.	7-49

* The privileged status column indicates whether the SYS call can be used only by a privileged user or by any user. A non-privileged user who attempts to call a privileged SYS function always receives the ILLEGAL SYS() USAGE error (ERR = 18). To avoid repetition in the documentation, error 18 is described for privileged calls only if it has a meaning different from non-privileged 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 non-privileged user, while the privileged user has a more powerful set.

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

(continued on next page)

Table 7-2: **FIP SYS Calls (by Sub-Function Code) (Cont.)**

Function Code(F0)	Privileged Status	Function Name	Page
-3	No	Get Monitor tables - Part I.	7-103
-2	Yes	Disable logins.	7-50
-1	Yes	Enable logins.	7-51
0	Yes	Create user account.	7-66
1	Yes	Delete user account.	7-68
2	Yes	Clean up a disk pack.	7-55
3	Yes	Disk pack and terminal status.	7-52
4	Yes	Login.	7-77
5	Yes	Logout.	7-78
6	Yes	Attach.	7-81
	Yes	Reattach.	7-82
7	Yes	Detach.	7-79
8	Yes	Change password/quota.	7-56
	Yes	Kill job.	7-57
	Yes	Disable terminal.	7-58
9	No	Return error messages.	7-33
10	Both	Assign/Reassign device or enter user logical.	7-34
11	No	Deassign device or remove user logical.	7-37
12	No	Deassign all devices.	7-39
13	Both	Zero a device.	7-40
14	Both	Read or read and reset accounting data.	7-89
15	No	Directory look up on index.	7-95
	No	Special magnetic tape directory look up.	7-97
16	Both	Set terminal characteristics.	7-69
17	No	Disk directory look up on filename.	7-99
	No	Disk wildcard directory look up.	7-100
18	Both	Send a message.	7-85
	Yes	Declaring a receiver and receiving a message.	7-83
	Yes	Remove from receive table.	7-86
19	Yes	Enable/Disable disk caching.	7-107
20	No	Date and time conversion.	7-147
21	Yes	System logical names.	7-125
22	Both	Message send/receive.	Ch. 8
23	Yes	Add/Remove system files.	7-129
24	Yes	Create a Job.	7-134
25	No	Wildcard PPN look up.	7-93
26	Both	Return job status.	7-140
--	Yes	PEEK function.	7-148

7.2.4 General Utility SYS Calls to FIP

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

(F0= -10)
(F0= -23)

7.2.4.1 Filename 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	The MSB (Most Significant Bits) of the file size as specified in the /FILESIZE:n (or /SIZE:n) file specification switch. If a non-zero number is returned, it indicates a large file. That is, the file size is greater than 65535 blocks.
3-4	Internal coding.
5-6	Project-programmer number (0 means the current account).
7-10	Filename 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; for files that are larger than 65535 blocks, the LSB (Least Significant Bits) of the file size.
15-16	The file cluster size given in the /CLUSTERSIZE:n file specification switch.
17-18	The value for MODE, if specified in the /MODE:n (or /RONLY) file specification switch, with the sign bit set.
19-20	The value for file position in the /POSITION:n switch, where n represents the device cluster number at which the first block of the file is placed.

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 non-zero and byte 22 contains the protection.
22	Protection code when byte 21 is non-zero.
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, the call returns the logical name in Radix-50 format in bytes 23 through 26. The Monitor does not translate the logical device name if the name is not currently assigned to a physical device or if the first character of the logical name string is an underscore (for example, OPEN "_KB:"). 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 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 filename string scan function determines specific file syntax information (for example, whether a given filename 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:

a. 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. Note that if the logical name string contains an underscore as the first character, the logical name is not translated. The STATUS variable is set for the device type found in the string scanned.

b. 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 wildcard character was found.

c. For a filename, the call validates the format and translates names into Radix–50 format. It also notes the presence of wildcard characters and detects file specification switches.

d. A filename extension is validated and translated into Radix–50 format. The call also notes the presence of wildcard characters.

e. 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.

f. 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 filename 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 FNP0$ 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 filename string scan call is the only function 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 filename 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 in the previous example.

Table 7-3: Filename 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	(S0% AND 8%) <> 0% (S0% AND 8%) = 0%	The /POSITION:n switch was specified. No /POSITION:n switch was found.
4-7	Reserved.	
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.

(continued on next page)

Table 7-3: **Filename String Scan Flag Word 1 (Cont.)**

Bit	Comparison	Meaning
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 was specified.
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 wildcard 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, does not correspond to any of the logical device assignments currently in effect or contains an underscore as the first character. The user program must test bits of flag word 2 for wildcard characters and device name found.

Table 7-4: **Filename String Scan Flag Word 2**

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

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

(continued on next page)

Table 7-4: Filename String Scan Flag Word 2 (Cont.)

Bit	Comparison	Meaning
5	(S1% AND 32%) <> 0%	Extension was an * character and is returned in bytes 11 and 12 as the Radix-50 representation of the string "???".
	(S1% AND 32%) = 0%	Extension was not an * character.
6	(S1% AND 64%) <> 0%	Extension contained at least one ? character.
	(S1% AND 64%) = 0%	Extension did not contain any ? characters.
7	(S1% AND 128%) <> 0%	A project-programmer number was found.
	(S1% AND 128%) = 0%	No project-programmer number was found (the following two comparisons return 0).
8*	(S1% AND 256%) <> 0%	Project number was an * character (that is the 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*	(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.
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 or contains an underscore; 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.

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

Since flag word 2 contains the high order byte of flag word 1 plus some additional 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   IF S1% AND 2048%    THEN  PRINT "ASSIGN'D PROTECTION USED"
220   IF S1% AND 4096%    THEN  PRINT "COLON (:) FOUND"
230   IF S1% AND 8192%    THEN  PRINT "DEVICE NAME FOUND"
240   IF S1% AND 16384%   THEN  PRINT "DEVICE NAME WAS LOGICAL"
250   IF S1%<0%           THEN  PRINT "DEVICE NAME NOT ASSIGN'D OR UNDERSCORE"
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? 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 apply:

1. When no device name is found in the string (no colon is found), the STATUS is random. This condition applies 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 or the logical name string begins with an underscore), STATUS is random. This condition applies 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, STATUS is set for the device. This condition applies 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 filename 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 (ERR=2) 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 filename string scan using CHR$(-23%) in place of CHR$(-10%) terminates without error on the following characters:

=	(equal sign)
/	(slash 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>" and RECOUNT equals 7. (The scan terminates on the comma between filenames.) 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/E 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 V7.0 SYSTEM #880
```

7.2.4.3 Assign/Reassign Device or Enter User Logical — Privileged and Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(10%), the assign/reassign and enter user logical code.
3–4	Not used.
5–6	For user logical assignment, the project-programmer number to be assigned. If bytes 7 through 10 are 0, these bytes contain the assignable project-programmer number (@). If bytes 7 through 10 contain a logical device name, these bytes contain the project-programmer number assigned to that logical device.
7–10	For assign device, bytes 7 through 10 must be 0. For reassign device, byte 7 is the job number to which the device is reassigned. Bytes 8 through 10 must be 0. For user logical assignment, bytes 7 through 10 contain the logical device name (in Radix-50 format) to be assigned.
11–12+	For assign/reassign device, either DOS or ANS (in Radix–50 format) to specify DOS or ANSI label format for the magtape drive. For user logical assignment, not used.
13–16	Not used.
17–18	For assign/reassign device, CVT%$(SWAP%(-32767%)), to assign a device that is currently assigned to another user. This usage is privileged and the target device must not be open. If this operation is not desired, bytes 17 and 18 are 0. For user logical assignment, not used.
19–20	Not used.
21	For user logical assignment, CHR$(255%) to enable protection code assignment (see Byte 22). For assign/reassign device, must be 0.
22	For user logical assignment, the protection code to be assigned. Byte 21 must be 255. For assign/reassign device, must be 0.
23–24+	Device name for assign/reassign and for user logical assignment.
25+	Unit number for assign/reassign and for user logical assignment.
26+	Unit number flag for assign/reassign and for user logical assignment.
27–30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

	Meaning	**ERR Value**
?ACCOUNT OR DEVICE IN USE	During a reassign call, the specified device is currently open or has an open file. During an assign call, more than four user logical assignments are made.	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	The device specified in bytes 23 through 26 exists on the system but the operation fails for one of the following reasons: 1. The device is currently reserved by another job (see bytes 17 and 18). 2. Ownership of the device requires privilege that the user does not have. 3. The device or its controller is disabled. 4. The device is a keyboard line for a pseudo keyboard use only.	8
?PROTECTION VIOLATION	A program with temporary privilege assigned a device, dropped its privilege, and attempted to reassign the device.	10
?ILLEGAL NUMBER	An attempt is made to transfer control to a nonexistent job. This error can only occur during a reassign call.	52

Discussion: The user logical assignment call is used to enter logical device names, logical project-programmer numbers, and default output protection codes. To assign a user logical device name, bytes 7 through 10 must contain the logical device name and bytes 23 through 26 must contain a physical device name and unit number. To assign a user logical project-programmer number, specify the number in bytes 5 and 6. To assign a user default protection code, specify the code in bytes 21 and 22.

The assign/reassign call uses bytes 17 and 18 to assign or reassign a device which is currently assigned. For the call to be successful, it must originate from a privileged account, the target device must not be open, and the current owner can not be performing a directory on that device.

The assign call reserves a physical device to a job* or transfers assignment of a currently owned device to another job. The actions are equivalent to the ASSIGN and REASSIGN Monitor commands. 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%.
```

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

7.2.4.4 Deassign a Device or Remove User Logical — Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(11%), the deassign device and remove user logical code.
3–4	Not used.
5–6	For user logical removal, the project-programmer number to be removed. For device deassignment, must be 0.
7–10	For user logical removal, the logical device name (in Radix–50 format) to be removed. Otherwise, must be 0.
11–20	Not used.
21	For user logical removal, CHR$(255%) to enable protection code removal. Otherwise, must be 0.
22	For user logical removal, the protection code to be removed. Byte 21 must be 255. Otherwise, must be 0.
23–24	For device deassignment, the device name to be deassigned. For user logical removal, must be 0.
25	For device deassignment, the device unit number to be deassigned. For user logical removal, must be 0.
26	For device deassignment, the unit number flag. For user logical removal, must be 0.
27–30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
?NOT A VALID DEVICE The device or device type specified in bytes 23 through 26 is not configured on the system. This error can occur only on device deassignment calls.	6

Discussion: The user logical removal call is used to deassign logical device names, logical project-programmer numbers, and default output protection codes. To deassign a logical device name, specify the name in bytes 7 through 10. To deassign a project-programmer number, specify the number in bytes 5 and 6. To deassign a user logical protection code, specify the code in bytes 21 and 22. Note that if these bytes do not contain specific deassignments, the call deassigns all user logical device name, PPN, and protection code assignments.

The device deassignment call performs the same action as the DEASSIGN system command described in the *RSTS/E System User's Guide*. For 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 — 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, in two Radix-50 words, for volume label (ANSI format magnetic tape only).
11–22	Not used.
23–24+	Device designator (disk, magnetic tape, 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 specified device is a magnetic tape with ANSI format, and the volume ID specified in bytes 7–10 is either missing or invalid.	2
?CAN'T FIND FILE OR ACCOUNT The account specified in bytes 5 and 6 does not exist on the device and unit number specified in bytes 23–26. That is, a device designator (e.g., DB1:) or logical name must be specified in bytes 23–24+ to cause this error; an SY: or default designator does not return this error.	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 8

The specified device 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.

?DEVICE NOT FILE STRUCTURED 30

The specified device does not allow access by file name.

Note 1: Only privileged users can specify an account other than their own account to be zeroed. Any values a non-privileged user specifies in bytes 5 and 6 are forced to the caller's own project-programmer number. Zeroes in bytes 5 and 6 indicate the project-programmer number of the calling program.

Note 2: When the zero a device SYS call is specified on magnetic tape or DECtape, the entire medium is zeroed without regard to any project-programmer number. On DECtape, the directory is cleared. Section A.3 describes what actions occur when magnetic tape 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.

a. The job must detach itself from its terminal. See the description of FIP code +7, Section 7.2.11.1.

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

c. The job must have DEASSIGNed any terminal which had been previously ASSIGNed to it. See the description of FIP code +11, Section 7.2.4.4.

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

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

a. The job can kill itself by means of FIP code +8, Section 7.2.5.11.

b. The job can find a free terminal (presumably the one from which it detached itself) and "force" into that terminal input buffer the character strings needed for logging into the system and attaching the job to the terminal. See the descriptions of FIP codes –4 (Section 7.2.5.5), +4 (Section 7.2.10.1), and +6 (Section 7.2.11.3).

c. The job can find a free terminal and use the REATTACH SYS call to attach itself to the terminal. See the description of FIP code +6 (Section 7.2.11.3).

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 BASIC-PLUS or is handled by the user's program itself. When the CTRL/C error condition is processed by BASIC-PLUS, 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.

(F0=–16)

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 met, 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 the *BASIC-PLUS Language Manual* for a description of the DATE$ function. Note that if D% in bytes 3 and 4 is 0%, no change is made to the current date.
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 the *BASIC-PLUS Language Manual* for a description of the TIME$ function. Note that if T% in bytes 5 and 6 is 0%, no change is made to the current time.
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 connected or 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.

Byte 4 of the data passed can contain the following values:

S%=-1%	Set "Data Terminal Ready" to permit a modem connected to a RSTS/E system to dial out. Should a connection not be established in 30 seconds, perform an automatic hang up of the dataset.
S%=0%	Hang up in two seconds.
S%=1-127	Hang up in one to 127 seconds.

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
?PROTECTION VIOLATION The job is not privileged or byte 3 contains an illegal KB: number.	10
?ILLEGAL BYTE COUNT FOR I/O An attempt was made to broadcast a zero-length message.	31

Discussion: The data broadcast is printed on the destination keyboard. The received message affects any output formatting being performed on the destination keyboard.

If the data is broadcast to a disabled keyboard line or to a hung up dataset line, control is returned to the program. No action is taken; no system error is generated.

Since the actual number of bytes broadcast depends upon the availability of small buffer space there may be occasions when the destination keyboard does not receive all of the bytes broadcast. Therefore, the program should test the value of the RECOUNT system variable to determine the number of characters not broadcast. If RECOUNT is not equal to zero, the program should then issue another broadcast call to transmit the remaining bytes. Refer to the *BASIC-PLUS Language Manual* for information on RECOUNT.

The following program segment illustrates how this SYS call and the RECOUNT variable can be used to ensure transmission of complete messages.

```
100   A$=SYS(CHR$(6%)+CHR$(-5%)+CHR$(N%)+M$)
110   IF RECOUNT <> 0% THEN
      M$=RIGHT(M$,LEN(M$)-RECOUNT+1%)
      \GOTO 100
```

While the RECOUNT variable is non-zero, the remainder of the string M$ is re-broadcast. When the complete message is broadcast and RECOUNT is 0, the program exits from the loop.

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
?PROTECTION VIOLATION The job is not privileged or byte 3 contains an illegal KB: number.	10
?ILLEGAL BYTE COUNT FOR I/O An attempt was made to force a zero-length string.	31

Discussion: The data forced is seen as input by the system.

If the data is forced to a disabled keyboard line or to a hung up dataset line, control is returned to the program. No action is taken; no system error is generated.

Since the actual number of bytes forced depends upon the availability of small buffer space, there may be occasions when the destination keyboard does not receive all of the bytes forced. Therefore, the program should test the value of the RECOUNT system variable to determine the number of characters not forced. If RECOUNT is not equal to zero, the program should issue another force call to transmit the remaining bytes. Refer to the *BASIC-PLUS Language Manual* for information on RECOUNT. Refer to Section 7.2.5.4 for an example of the use of RECOUNT.

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 logins allowed.
4-30	Not used.

Possible Errors: No errors are possible.

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

* The number of jobs that can log into a RSTS/E system is limited by the swapping space available, the JOB MAX set at system start up, and the set maximum number of logins. However, console terminal KB0: is a special terminal and can log in regardless of the set login maximum, provided that swapping space and JOB MAX permit. The system manager can install a patch that changes the number of the special keyboard from KB0: to KBn:.

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, Section 7.2.8.
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: The following data is returned for a disk pack or cartridge 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 ONLCLN 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 equivalent to FIP call 16 discussed in Section 7.2.8. For a discussion of disk management on RSTS/E, see 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 specified device 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, 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. Because the use of this SYS call to clean a disk ties up system resources (FIP) for a considerable amount of time, the use of the on-line clean program (ONLCLN) is recommended. See 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. If N% is zero, the quota is unlimited (see Byte 21).
15–20	Not used.
21	CHR$(255%) to change the quota (see bytes 13–14). Zero to indicate that no change is to be made to the quota.
22	Not used.
23–24+	Device name. If no device name is specified, SY: is used.
25+	Unit number.
26+	Unit number flag.
27	Not used.
28	Must be CHR$(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, and can be executed under program control by any (privileged) program to terminate the job.

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.

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.*

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	Not used.
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 can 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.

Note that because of the manner in which RSTS/E interprets CCL commands, ensure that similar commands are defined in the correct order. That is, MACRO must be defined before MAC.

7.2.6 Job Scheduling SYS Calls to FIP

(F0= -13)

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 31K 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.

Note that the Monitor uses 256 queues to schedule jobs and each queue is polled in sequential priority order. The Monitor chooses the highest priority runnable job as the job to be run. Thus, it is possible for a high priority compute-bound job to monopolize system resources.

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 Section 1.2.) 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

(F0=0)

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
?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 the *RSTS/E System Manager's Guide* 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 specified account 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 by means of 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 and Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(16%), the set terminal characteristics code; if this byte is 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 (non-privileged); or is the keyboard number of the terminal to alter (privileged).
5	CHR$(N%), where N% is 0% for no change or is the terminal width plus 1. The call sets the number of characters for each line to N% minus 1, where N% is in the range of 2% and 255%. The WIDTH n command sets this byte.
6	CHR$(N%), where N% is: 0 for no change. 128 to enable the 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 the 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 LC OUTPUT command sets this characteristic.

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

0	for no change.
128	to have the 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	the terminal has the requisite hardware to respond to XON and XOFF characters. The terminal stops sending characters when it receives a CHR$(19%) character (XOFF) and resumes sending characters when it receives a CHR$(17%) character (XON). The XON command sets this characteristic.

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

0	for no change.
128	to have characters that are typed at the terminal sent to the computer only. The computer echoes (transmits back to the terminal) the characters 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 the terminal does not have features of a video display terminal. The NO SCOPE command sets this characteristic.

255 the 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 the system treats certain characters that it receives 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%).

The NO LC INPUT command sets this characteristic.

255 the terminal transmits the full ASCII character set and the 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.

The LC INPUT command sets this characteristic.

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

0	for no change.
1	to have no fill factor for the terminal. The NO FILL command sets this characteristic.
n	set fill factor of the terminal to N% minus 1. The FILL n command determines this value.
255	set fill factor for an LA30S (serial) DECwriter. The FILL LA30S command sets this characteristic.

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

0	for no change.
n	the internal speed value to determine the baud rate at which the terminal receives characters. If byte 16 is 0, this value also determines the transmit (output) baud rate.
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. The NO PARITY command sets this value.
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 the value 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.
255	2741-type terminal. Byte 14 must also be 255.

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

0 for no change.

128 the terminal ignores the synchronization protocol that is described in the following 255 value. The NO STALL command sets this characteristic.

255 the terminal obeys the synchronization protocol as follows:

a. the computer stops sending characters if the terminal transmits a CHR$(19%) character (XOFF, or the CTRL/S combination).
b. the computer resumes sending characters when the terminal transmits a CHR$(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	the system prints a control character 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	the system treats control characters as such. The NO UPARROW command sets this characteristic.

19 No effect.

20 CHR$(N%), where:

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.

Note that this byte applies only to those interfaces that support DATA/STOP/PARITY features. When this byte is used on applicable interfaces, it overrides the setting of byte 15. Also, when byte 20 is used, byte 14 must be set to a non-zero value.

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

0 for no change.

255 set the ring list entry for a terminal (which is attached to a modem-controlled line interface) to default to permanent characteristics when the modem is answered. The /RING option with a TTYSET KBn: command determines this value.

Note that this byte is always forced to 0 for non-privileged users.

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 (in the range 1 to 127). If the character has a special meaning (for example, horizontal tab or the CTRL/Z combination), the private delimiter usage has higher precedence. The delimiter cannot be used with INPUT, INPUT LINE, or MAT INPUT statements. See Table 4-3 for a discussion of delimiters.

The DELIMITER x and DELIMITER "x" commands set this value.

24	CHR$(N%), where N% is:
	0 for no change.
	128 the terminal in use does not have the ESC key which generates CHR$(27%). Therefore, translate ALT MODE CHR$(125%) and PREFIX CHR$(126%) to CHR$(27%). The NO ESC command sets this value.
	255 the terminal in use has an ESC key which generates CHR$(27%). Therefore, do not translate CHR$(125%) and CHR$(126%) but treat them as their ASCII characters } and ~. The ESC command sets this value.
25	CHR$(N%), where N% is:
	0 for no change.
	128 to disable the CTRL/R and CTRL/T facilities.
	255 to enable the CTRL/R and CTRL/T facilities.
	Note that the NO CTRL/R and CTRL/R commands disable and enable the CTRL/R facility, which retypes your terminal's pending input buffer. When you type CTRL/T, a status report of the current keyboard is produced (unless this byte is set to 128 to disable CTRL/T).
26	CHR$(N%), where N% is:
	0 for no change.
	128 to define XOFF/XON processing such that the keyboard resumes typeout and echo only after XON or CTRL/C is typed.
	255 to define XOFF/XON processing such that the keyboard resumes typeout and echo when any character is typed after XON. The RESUME ANY and RESUME CTRL/C commands set this characteristic.

Data Returned:

Byte(s)	Meaning
5–26	These bytes return values that define the current keyboard characteristics; with two exceptions, as follows: Byte 20 is always returned as 0. Byte 19 returns a code that defines the type of interface for the line, where: 0 is KL11, DL11A, DL11B 2 is DC11 4 is DL11C, DL11D 6 is DL11E 8 is pseudo keyboard 10 is DJ11 12 is DH11 14 is DZ11

Possible Errors:

	Meaning	ERR Value
?ILLEGAL SYS() USAGE	One of three possible error conditions can cause this error; 1. The keyboard number specified in byte 4 of the call is out of the range of valid keyboard numbers. 2. The current keyboard is specified (byte 4=255) but is detached. 3. A non-privileged user specified an actual keyboard number.	18

Discussion: If the terminal specified by the keyboard number in byte 4 of the call is disabled (as a result of the system initialization procedure or of executing the disable terminal SYS call) the call is not executed by the system and no error occurs.

Use this call to determine the current keyboard characteristics and then to make changes to those characteristics.

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*.
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-78 30-Sep-78  03:13 PM

Ready
```

* 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–78, and the date of last access to 21–Jul–78, as shown on the partial directory listing following the program.

```
10      D% = 8203%
        !21-JUL-78 IS (203) + ((1978-1970)*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 LAST ACCESS
400     M%(6%) = D% AND 255%
        \M%(7%) = SWAP%(D%) AND 255%
        !SET UP THE DATE OF CREATION
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-78 21-Jul-78 12:00 M
TEMP20.TMP      1       60      30-Sep-78 30-Sep-78 03:14 PM

Ready
```

7.2.10 LOGIN and LOGOUT SYS Calls

(F0=4)

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]; it 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. Note that this call does not immediately terminate the job; the Monitor kills a logged out job when the program the job is running terminates. To perform all of the LOGOUT clean-up functions and cause the job to terminate itself, use the kill job FIP call (F0=8) described in Section 7.2.5.11.

7.2.11 Detach, Attach, and Reattach SYS Calls

(F0=7)

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 system associates the activated job with both the terminal at which the user is typing and the account number which was used for system identification. 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 you desire to attach a detached job to a terminal, 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.*) Because 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. You log into the system again with the same account number and reattaches to the job you were previously running.

If the detached job has its console terminal open on some non-zero channel, it can perform I/O on the keyboard from which it is detached. It must use a non-zero 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. A detached job which performs I/O on the detached keyboard on a non-zero 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 non-zero I/O channel. The detached job can continue to perform I/O on the non-zero 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 non-zero 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 non-zero 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 source 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 – Privileged and Not Privileged

NOTE

This call exists only for compatibility with previous implementations of message processing and its use is not recommended. The following sections contain a summary of this call's format. For a complete description of this call refer to previous editions of the *RSTS/E Programming Manual.* For a complete description of the recommended call for message processing, refer to Chapter 8.

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 sender's job number times 2.
6	Must be CHR$(0%).
7–8	Project-programmer number of the sending 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
?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 the Monitor. Error is returned to the program when the 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
?NO BUFFER SPACE AVAILABLE When the job attempted to declare itself as a receiving job, there were no small buffers available for the declaration arguments. Because small buffers are dynamically allocated, wait and then retry the operation.	32

7.2.12.2 Send a Message — Privileged and Not 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$(-1%); this value indicates that the call is a request to send a message.
4	CHR$(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 identification list 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. Sending a message requires a small buffer and one is not available.	32

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.

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 you desire a byte change, read the word (using the PEEK function, see Section 7.3), 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.* 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.

* The number of jobs that can log into a RSTS/E system is limited by the swapping space available, the JOB MAX set at system start up, and the set maximum number of logins. However, console terminal KB0: is a special terminal and can log in regardless of the set login maximum, provided that swapping space and JOB MAX permit. The system manager can install a patch that changes the number of the special keyboard from KB0: to KBn:.

7.2.15 Accounting Information

(F0=14)

7.2.15.1 Read or Read and Reset Accounting Data — Privileged and Not Privileged

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 (maximum number returned is 65535 blocks; thus if there are more than 65535 blocks, only 65535 is returned).
7-8	Project-programmer number of the account read.

9–12	Password of the account read; in Radix–50 format. This data is returned only if the caller is privileged.
13–14	Low order word (16 bits) of the CPU time (in tenths of seconds) used by the account.
15–16	Connect time (in minutes) used by the account.
17–18	Low order word (16 bits) of kilo-core ticks used by the account.
19–20	Device time (in minutes) used by the account.
21–22	High order bits for CPU time and kilo-core ticks. See the Discussion for an explanation of how the values are stored.
23–26	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.

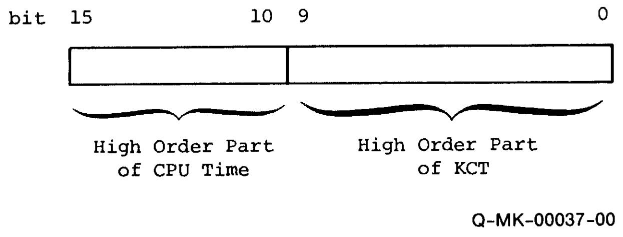
Possible Errors:

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 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 program in the *RSTS/E System Manager's Guide* 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 non-privileged users through the QUE command. It is desirable that the calling user be charged for the time the SPOOL program accumulates processing the job. The SPOOL program 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.

7.2.15.3 Wildcard PPN Look Up — Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(25%), the code for account number look up on index.
3–4	CHR$(N%)+CHR$(SWAP%(N%)), where N% is the index of the desired project-programmer number. If N% is 0, the call returns the project-programmer number of the first account on the disk that matches the wildcard specification. If N% is non-zero, the call returns the project-programmer number of the N+1 account on the specified disk that matches the wildcard specification.
5–6	The desired project-programmer number. A specification of 255 in either field represents a wildcard. A specification of 0 for N% in bytes 3–4 and a non–255 value in bytes 5-6 (no wildcard) verifies the existence of the specified account on the disk. If both fields are non-zero and neither field is 255, the call returns error 5.
7–22	Not used.
23–24+	The device name, which must be a disk. If bytes 23 and 24 are both 0, SY0: (the system disk, not the entire public structure) is used.
25+	The device unit number.
26+	The unit number flag. If byte 26 is 0, SY0: (the system disk) is used.

Data Returned:

Byte(s)	Meaning
3–4	Internal code.
5–6+	The project-programmer number located by the call.
7–30	Same as passed.

Possible Errors:

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT The specified device in bytes 23–24+ is not a disk or no match exists for the specified index value in bytes 3–4 (see also bytes 5–6).	5
?DISK PACK IS LOCKED OUT The disk is in the locked state. Execute the call under a privileged account to override this condition.	22

Discussion: This call allows you to specify a wildcard account number and increment an index value to determine a matching project-programmer number. The wildcard account specification is as returned from the filename string scan call (see Section 7.2.4.1).

7.2.16 Directory Look Up

The SYS function calls described in this section look up filenames 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 following format:

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. For magnetic tape, these bytes return the value passed.
7–10	Filename 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 magnetic tape look up described in Section 7.2.16.2); LSB (Least Significant Bits) for files larger than 65535 blocks.
15	Protection code of the file.
16	The most significant bits of the file size (MSB size). If a non-zero number is returned, it indicates a large file. That is, the file size is greater than 65535 blocks.
17–18	For disk, the date of last access*; for DECtape and magnetic tape, returns the date of creation.
19–20	The date of creation for disk.
21–22	The time of creation for disk; for DOS magnetic tape, the project-programmer number.
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	File is placed.
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.

* 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 magnetic tape, N% must be 0 to rewind the tape before reading the first file. See Section 7.2.16.2 for a description of magnetic tape 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 magnetic tape, 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 in 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 magnetic tape, 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 Magnetic Tape 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 magnetic tape, 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 magnetic tape directory look up.
7–22	Not used.
23–24	MT, MS, or MM.
25+	Unit number.
26+	Unit number flag.
27–30	Not used.

Data Returned: See introductory material in 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 magnetic tape unit results in the following actions by the Monitor:

a. reads one record from the tape (a label record).*
b. spaces the tape forward to the next end-of-file record and calculates the number of records in the file.
c. returns the directory information if the account number of the file matches the one specified in the call or if both bytes in the account specification in the call are zero.

* 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 a, the tape must be positioned immediately before a label record. Otherwise, an error is generated or incorrect 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 magnetic tape 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 a.
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 a.

Note that the required rewind operations consume time. To avoid the rewind operations, the application program can execute the special magnetic tape directory look up call and perform certain actions. By specifying both bytes 5 and 6 as CHR$(255%) in the call, the program causes the following actions by the monitor:

a. reads from the tape a record which must be a label record.
b. backspaces one record which leaves the tape in a position to read the label record again.
c. returns the directory information (except for file length) to the program.

To take advantage of this special action, the program can perform the following actions:

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 filename 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 tape mark (see Section 2.8.4).
3. after processing the required file, execute a CLOSE statement to position the tape at the tape mark 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 Filename — Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(17%), the disk directory look up by filename and disk wildcard directory look up code. See Section 7.2.16.4 for a description of the wildcard look up 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+	Filename 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 in 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	The file identification index is returned.

Possible Errors:

Meaning	ERR Value
?ILLEGAL FILE NAME Filename 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 Wildcard 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 wildcard specification. If I% is x, return the data for the x + 1 file which matches the wildcard specification.
7-10+	Radix-50 representation of a wildcard filename specification where an * character can replace the filename or a ? character can replace any character in the filename. Used with the extension in bytes 11 and 12 to create a wildcard file specification.
11-12+	Radix-50 representation of a wildcard extension specification were an * character can replace the extension or a ? character can replace any character in the extension. Used with the filename in bytes 7 through 10 to create a wildcard file specification.

Data Returned: See introductory material in Section 7.2.16.

Possible Errors:

Meaning	ERR Value
?ILLEGAL FILE NAME No filename 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 wildcard specification and to increment an index value to gain directory information for all occurrences of files matching the wildcard specification. The following are typical wildcard 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 wildcard 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 filename 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 magnetic tape, 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 filename blocks for indices numbered 0 through N–1 before it reads the filename block for index number N. The process is the only one possible because the index value has no other relationship to the actual disk address of the filename block. Note that the index scheme for SYS call 16 (and 25) differs from that of SYS call 14.

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 filename 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:).

(F0=-3)

(F0=-12)

(F0=-8)

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. The calls 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.
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	(UNTOWN) - The table of unit owners (low byte) and options (high byte).

Possible Errors: No errors are possible.

7.2.17.2 Get Monitor Tables - Part I — 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 of physical device addresses.
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	If large file capability is present on the system, these bytes contain a pointer to the FCBLST table. FCBLST contains a word, for each generated unit, that is the root of a linked list of file control blocks (FCBs) for open files on that unit. If large file capability is not present on the system, these bytes return zero.

●

Possible Errors: No errors are possible.

7.2.17.3 Get Monitor Tables - Part II — 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.

For an alternative to this call, see the STATUS variable described in 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 cache code.
3	CHR$(N%), where N% is: 0 to enable directory and data caching. In addition to this byte, data caching requires a value setting in byte 11. Note that bytes 4 through 12 are used only if this byte equals 0. 128 to return the current caching parameters. A 128 value in this byte does not enable or disable data caching. 1 to disable all caching.
4	CHR$(C%), where C% is the cache cluster size. If C% is 0, the current cluster size is used (see Discussion). Cache cluster size can be specified as 1,2,4, or 8 blocks. If C% is greater than 8, 8 is used.
5–6	CHR$ (L%)+CHR$(SWAP%(L%)), where L% sets a limit on the total number of cache clusters that can be used. If L% is 0, the current limit is used (see Discussion). If L% is non-zero, it specifies an upper limit on the number of clusters in the cache. Note that if the amount of XBUF available to the cache is less than L%, the cache will not exceed XBUF.
7–8	CHR$(D%)+CHR$(SWAP%(D%)), where D% sets a limit on the total number of cache clusters allocated for directory caching. If D% is 0, the current limit is used (see Discussion). If D% is non-zero, it specifies an upper limit for the number of clusters in the cache that are available for directory caching. Note that the number of clusters allocated for directory caching during a particular operation can be less than D%.
9–10	CHR$(U%)+CHR$(SWAP%(U%)), where U% sets a limit on the total number of cache clusters allocated to user data caching. If U% is non-zero, it specifies an upper limit for the number of clusters in the cache that are available for user data caching. Note that the number of clusters allocated for data caching during a particular operation can be less than U%.
11	CHR$(E%), where E% modifies the enabling/disabling of data caching as follows: E%=0 use the current setting. E%=1 enable data caching as specified in file OPEN MODE or UFD setting (see Section 1.4.9). E%=128 disable all data caching. E%=64 cache all data transfers regardless of file OPEN MODE or UFD setting.
12	CHR$(M%), where M% controls the cache's use of the small buffer pool, as follows: M%=0 use the current setting. M%=1 use the small buffer pool. M%=128 do not use small buffer pool.
13–30	Not used.

Data Returned:

Byte(s)	Meaning
1–2	Internal coding.
3	Current cache setting and available options: 0 Cache disabled, user data caching is not available. 1 Cache enabled, user data caching is not available. 128 Cache disabled, user data caching is available. 129 Cache enabled, user data caching is available.
4–12	Current settings of cache parameters, as described for passed data. Note that these bytes have meaning only if the system manager has installed disk caching during system generation.
13–30	Not used.

Possible Errors:

Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE All of the clusters allotted to the cache are in use.	3
?NO ROOM FOR USER ON DEVICE An attempt was made to enable data caching without sufficient XBUF space allocated to the cache. The system manager must allocate at least 2K words of memory to XBUF for caching.	4
?DEVICE NOT AVAILABLE An attempt was made to change the cache cluster size (see Byte 4) while a cached file disk transfer was in progress. Retry the operation.	8
?PROTECTION VIOLATION The current user is not privileged.	10
?MISSING SPECIAL FEATURE Caching was not enabled for the system during system generation.	66

Discussion: Bytes 1,2, and 3 of this call enable or disable the FIP buffering module that controls directory caching. The ENABLE CACHE and DISABLE CACHE commands of UTILTY use these bytes.

If user data caching was installed on the system during system generation, bytes 1 through 12 enable or disable user data caching and set the parameters of the cache. The system manager defines the total size of XBUF during system generation and some portion of this space is, in turn, used by the cache. The disk caching SYS call defines the size of the directory portion and data portion of the cache. The sizes defined in this call set upper limits, not fixed sizes. Thus, if a 40K word XBUF were defined at system generation, the SYS call could define the directory and data portions of the cache as 25K each. That is, data can use the space in the cache up to a maximum of 25K, which leaves a minimum of 15K for the directory. The reverse is also true. In this manner, data and directory caching are guaranteed a minimum allocation and are allowed to overlap, which permits the cache to dynamically adjust to system and program requirements.

This SYS call is also used to limit the size of the total cache. Because XBUF is used by the cache and DECnet/E, limiting the cache guarantees that space is always available in XBUF for DECnet/E. Note that the amount of memory allocated to the cache is freed for other use when caching is not being performed.

Byte 4 of the call sets the cache cluster size. This parameter controls the number of contiguous blocks that are copied frm the disk to the cache whenever a file or directory is cached (see Section 1.4.9). The cache cluster size should be small enough to contain a reasonable number of clusters, but large enough to reduce the number of disk accesses. That is, you must anticipate data requests and ensure that the cache is equal to the file cluster size of the most often accessed file. If you specify a cache cluster size of 1, only random caching (see Section 1.4.9) is allowed. Refer to the *RSTS/E System Manager's Guide* for cache cluster size guidelines.

Note that the parameters for cache cluster size and cluster allocation (Bytes 4 through 10) have default settings at system startup. The default settings are; a cache cluster size of 4 and no limits on directory, data, or total cache size. The system manager can reset these defaults with an INIT option as described in the *RSTS/E System Generation Manual.*

7.2.19 Run-Time System and Resident Library Control

A privileged user can conduct time-sharing operations with an auxiliary run-time system (RSX, RT-11, etc.) supplied by DIGITAL. This section describes SYS system function calls -17 and -18, used to add, remove, load, or unload auxiliary run-time systems. The section also describes the use of SYS call -18 to perform these operations on resident libraries. These calls are used by the UTILTY system program described in the *RSTS/E System Manager's Guide.*

(F0= -17)

7.2.19.1 Associate a Run-Time System with a File — Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-17%), the associate 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.

With the exception of files that are greater than 65535 blocks (see Discussion in Section 7.2.24), 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.
3	CHR$(0%), to add a run-time system.
4	Not used.
5-6+	Project-programmer number of the file to add; if none is specified, [0,1] is the default.
7-10+	Run-time system name in Radix-50 format.
11-12	CHR$(A%)+CHR$(SWAP%(A%)) where A% is the 1K-word section of memory at which this run-time system is to be loaded. The numbering begins at 0 and ends at n-1 (where n is the total number of 1K-word sections of memory on the system). If A% is 0%, the Monitor decides where to load the run-time system. Note that when a run-time system with read/write characteristics (see Bytes 19-20) is added, the Monitor cannot decide where to load it. Therefore, if read/write characteristics are desired, you must specify the run-time system load address.
13-14	Maximum allowed user image size, in K words (the P.SIZE symbol).
15-16	Minimum allowed user image size, in K words (the P.MSIZ symbol).
17	CHR$(P%) where P% is the position in the linked list of run-time system (RTS) 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 non-zero 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$(S%) where S% is the stay flag. If S% is 128% (the high bit is set), this RTS is kept permanently resident. (The usage count remains non-zero.) If S% is 0%, the memory occupied by this RTS can be released as user job space whenever the usage count of the RTS goes to 0. Note that if S% is 128%, the load address must be specified in Bytes 11-12.
19-20	CHR$(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. A load address must be specified in Bytes 11-12.
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 < code < 255).

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 non-zero, the Monitor uses that as the load address. If A% is non-zero 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 non-zero and the add load address was also non-zero, 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
?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 portion of memory occupied by the run-time system. The memory is made available as user job space.

7.2.19.6 Add a Resident Library — Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-18%), the resident library manipulation code.
3	CHR$(16%), add a resident library.
4	Not used.
5-6+	The project-programmer number of the file to add; if none is specified, [0,1] is the default.
7-10+	The resident library name in Radix-50 format.
11-12	CHR$(A%)+CHR$(SWAP%(A%)), where A% is the 1K-word section of memory at which the resident library is to be loaded. The numbering begins at the first available 1K-work section and ends at n-1 (where n is the total number of of 1K-word sections of memory on the system). A% must be specified and cannot be zero.
13-17	Not used.
18	CHR$(S%), where S% is the stay flag. If S% is 128% (the high bit set), the resident library is made permanently resident. If S% is 0%, the memory occupied by the library can be freed for user space whenever the usage count of the library is zero (no active task is accessing the library).
19-20	CHR$(F%)+CHR$(SWAP%(F%)), where F% is the flag word that defines the characteristics of the library. Only the high byte is used for flag bits. F% is the sum of the bits set, as follows:

F%=256%	Not used.
F%=512%	The resident library is available to only one user. It is not shared by multiple users.
F%=1024%	The resident library allows read/write access to its memory, rather than read only access.
F%=2048%	The resident library does not record errors in its code in the system error log.
F%=4096%	The resident library is immediately removed from memory when its usage count equals zero.
F%=8192%	Not used.
F%=16384%	Not used.
F%=32767%+1%	Not used.

21-22+	Protection code for the installed resident library. To specify a protection code; byte 21 is non-zero and byte 22 contains the desired protection. To accept the default protection, byte 21 is 0. The default protection is <42>, which means that the Monitor grants read access to all users but denies write access.
23-24+	The name of the disk device on which the resident library is to be 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 You specified an address in bytes 11 and 12 that would cause the Monitor to load the library in such a manner that memory would be fragmented and a swapping violation would occur. Refer to the *RSTS/E System Generation Manual* for guidelines on avoiding memory fragmentation.	4
?CAN'T FIND FILE OR ACCOUNT You specified a filename in bytes 7 through 10 that cannot be found in the account specified in bytes 5 and 6 on the device specified in bytes 23 through 26. Ensure that the filename you specify has a .LIB extension and is located in the specified account and device.	5
?PROTECTION VIOLATION The file you wish to add is in improper format. For example, this error occurs if you specify a file that is not contiguous or has illegal entries in the SIL index.	10
?NAME OR ACCOUNT NOW EXISTS You specified the filename of a resident library that already exists.	16
?ILLEGAL BYTE COUNT OF I/O You did not specify a load address in bytes 11 and 12 or the address you specified is not available. Refer to the memory status report of a display program to determine an available range of memory.	31
?NO BUFFER SPACE AVAILABLE A small buffer is required for the description block of an added resident library. This error is returned if a small buffer is not available.	32

Discussion: The Monitor SYS call to add a resident library is privileged and allows you to add a specified library to the Monitor's list of resident libraries. This call is similar to that used to add a run-time system except that you must specify a load address for the library. That is, unlike run-time systems, the Monitor does not automatically decide where to load a resident library. Also, the library file does not have to reside in account [0,1]; however, the file extension must be .LIB.

For additional information on the creation and use of resident libraries, refer to the *RSTS/E Task Builder Reference Manual.*

7.2.19.7 Remove a Resident Library — Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(−18%), the resident library manipulation code.
3	CHR$(20%), remove a resident library.
4–6	Not used.
7–10+	The resident library 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 You attempted to remove a library that is being loaded into memory or is in use by the currently running job. A resident library cannot be removed while a job is still attached to it.	3
?CAN'T FIND FILE OR ACCOUNT You specified a resident library name in bytes 7 through 10 that is not currently defined.	5

Discussion: The Monitor SYS call to remove a resident library is privileged and allows you to remove the library from physical memory, delete the Monitor structure that defines the library, and close the library file.

7.2.19.8 Load a Resident Library — Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-18%), the resident library manipulation code.
3	CHR$(18%), load a resident library.
4–6	Not used.
7–10+	The resident library name in Radix-50 format.
11–12	CHR$(A%)+CHR$(SWAP%(A%)), where A% is the 1K-work section of memory at which the library is to be loaded. The numbering begins at the first available 1K-word section and ends at n–1 (where n is the total number of 1K-word sections of memory on the system). If A% is 0%, the Monitor uses the load address that was initially set for the library when it was added. If A% is non-zero, the value specified replaces that initially set for the library.
13–17	Not used.
18	CHR$(S%), where S% is the stay flag. If S% is 128% (the high bit set), the library is made permanently resident (i.e., the usage count is always non-zero). If S% is 0%, the memory occupied by the library is freed for user space whenever the usage count is zero.
19–30	Not used.

Data returned: No meaningful data is returned.

Possible errors:

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE You specified an address in bytes 11 and 12 that would cause the Monitor to load the library in a position where memory would be fragmented and a swapping violation would occur. Refer to the *RSTS/E System Generation Manual* for guidelines on assigning and allocating memory to avoid fragmentation.	4

?CAN'T FIND FILE OR ACCOUNT		5
	You specified a resident library name in bytes 7 through 10 that is not currently defined.	
?ILLEGAL BYTE COUNT FOR I/O		31
	The load address was not specified in bytes 11 and 12 or the address specified is not available. Refer to the memory status report of a display program to determine an available range of memory.	

Discussion: The Monitor SYS call to load a resident library is privileged and allows you to specify a position in memory for the library. That is, you can use this call to move a resident library from its current position in memory to a position of your choice. Note that if you wish to move a library to a permanent position in memory, you must set the stay bit (see format description). The stay bit ensures that the library will remain in the specified location for the duration of the current time-sharing session.

7.2.19.9 Unload a Resident Library — Privileged

Data passed:

Bytes(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-18%), the resident library manipulation code.
3	CHR$(22%), to unload a resident library.
4-6	Not used.
7-10+	The resident library 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 You attempted to unload a resident library that is in the process of being loaded or is in use by the currently running job. A library cannot be unloaded while a job is still attached to it.	3
?CAN'T FIND FILE OR ACCOUNT You specified a resident library name in bytes 7 through 10 that is not currently defined.	5

Discussion: The Monitor SYS call to unload a resident library is privileged and allows you to remove the library from memory and free that portion of memory for use by other jobs.

7.2.20 Read and Write Attributes

Certain PDP-11 record organizations, such as RMS-11, 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. For a description of file attributes, refer to the *RSTS/E System User's Guide.*

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, all bytes 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

* To determine the number of attributes returned, scan backwards from byte 26 (in words) to find the first word that is not null. Then calculate the number of attributes returned. If all the words are null, no attributes were returned.

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

NOTE

DIGITAL supplied software is dependent on file attribute data as defined by the system. User-written software must not write attribute data that conflicts with system defined attribute data.

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 non-disk 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.* 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 was 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 specification SCRACH:OTHER.DAT refers 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.

* The capability to define system logical names for non-disk 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 generates the ?NO ROOM FOR USER ON DEVICE error (ERR=4).

7.2.21.1 Add New 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$(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 non-alphanumeric 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 non-alphanumeric 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 non-alphanumeric 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 non-alphanumeric characters.	2
?ACCOUNT OR DEVICE IN USE A swapping file is being added to a non-file 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 should be at least 64 blocks and the error file at least 16 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.	4
?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 non-file 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 file 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 INIT or the 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 file or specifies a file to be added as a swapping file, 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 non-system (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 the contiguous space at the proper size. If a fixed head disk is to be added as a swapping file, 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$(N%) where N% designates the file to remove as follows: 0 swapping file 0. 1 swapping file 1. 2 illegal - generates error. 3 swapping file 3. 4 overlay file. 5 error message file.
4	CHR$(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 18

The number specified in byte 3 is either 2 or is greater than 5. The swapping file for file 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.

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 file for that device, dynamically add another device or file to replace the disk as a swapping file 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.

7.2.23 Create a Job — Privileged

Data passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(24%), the job creation code.
3	Not used, must be 0.
4	CHR$(N%) where N% is: 0% create the job only if logins are enabled. 128% create the job even if logins are disabled.
5–6+	The project-programmer number of the program to be run.
7–10+	Program name of the program to be run in Radix-50 format.
11–12+	Extension of the program to be run in Radix-50 format.
13–22	Ten bytes of information; placed into the created job's Core Common area. Note that an eleventh byte is appended that contains the job number times 2 of the job that executed the SYS call.
23–26+	The device name and unit number of the program to be run.
27–28	The parameter word to be passed to the program to be run. The parameter word has exactly the same format and functions as the CCL command parameter word. Note that for jobs created under the BASIC-PLUS Run-Time System, the parameter word equals the program line number to which control is transferred when the job runs.
29–30	Not used.

Data Returned:

Byte(s)	Meaning
3	The job number times 2 of the job just created.

Possible Errors:

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE The new job can not be created. Probable causes are: a. further logins are disabled and byte 4 is 0. b. the system's job or swap slots are (currently) full.	4

Discussion: To create a job, specify the complete filename of the program the created job is to run. The specification must include the project-programmer number; there is no default.

The program must be compiled (executable). You cannot run a source program because the run-time system cannot create the required work files.

The system default run-time system must be one of the following DIGITAL supplied run-time systems: BASIC-PLUS, BASIC-PLUS-2 (BP2COM), or RSX. Note that this restriction applies only to the system default run-time system; the creating job and the created job can run under control of any run-time system.

The created job is not logged in when it runs. Therefore, all files accessed by the job must be completely specified, including project-programmer numbers.

The created job runs logged out, which means that if the job fails and exits, the system kills the job. To cause the created job to hibernate on failure rather than exit, have it execute the login SYS call (see Section 7.2.10.1).

The created job runs at priority zero, which in the case of an infinite loop, may seriously degrade system performance. To avoid this condition, cause the created job to reset its priority on execution.

7.2.24 File Utility Functions – Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-26%), the file utility code.
3	CHR$(N%), where N% is the internal channel number (in the range of 1 to 12) on which the file is open. If N% is 0, specify the target file by project-programmer number, filename, and extension in bytes 5 through 12.
4	CHR$(F%), where F% specifies the file utility function. The function F% is one (or the sum) of the following codes:

1%	set or reset the file's placed bit (cannot be used with code 8%). See byte 15.
2%	modify code 16% to return 0 as the device cluster number (DCN) if the file's placed bit is not set.
4%	change the file's backup statistics.
8%	change the file's run-time system name field.
16%	return the file's retreival information. That is, 16% causes the Monitor to map the virtual block number (VBN) of the file into the disk DCN. This code can be used to obtain an existing file's DCN in order to place a new file near it (see Discussion).
32%	unset the file's contiguous bit. This code allows you to extend a contiguous file, however, the file is made non-contiguous.
64%	enable/disable sequential mode caching if the file is cached (cannot be used with code 8%). See byte 15.
128%	enable/disable data caching on the file (cannot be used with code 8%). See byte 15.

5–6+	If N% in byte 3 is 0, specify the project-programmer number of the file you wish to modify. If N% is non-zero, these bytes are ignored.
7–10+	If N% in byte 3 is 0, specify the filename (in Radix–50 format) of the file you wish to modify. If N% is non-zero, these bytes are ignored.
11–12+	If N% in byte 3 is 0, specify the file extension (in Radix–50 format) of the file you wish to modify. If N% is non-zero, these bytes are ignored.

13–16	The specifications that you make in these bytes are dependent on the function code specified in byte 4, as follows: If byte 4 AND 8%<>0%, then bytes 13 through 16 contain the new run-time system name field in Radix-50 format. Otherwise; If byte 4 AND 16%<>0%, then bytes 13 and 14 contain the low order word of the VBN you wish to locate, byte 15 contains 0% or is used by another operation, and byte 16 contains the high order word of the VBN you wish to locate. If byte 4 AND 1%+64%+128%, then bytes 13, 14 and 16 contain zeros or are used by another operation, byte 15 contains flags for the following operations: 2% new value for the placed bit if byte 4 AND 1%<>0%. 4% new value for sequential bit if byte 4 AND 64%<>0%. 128% new value for cached bit if byte 4 AND 128%<>0%.
17–18	If the change file backup statistics function is selected in byte 4 (code 4), these bytes specify a new date of last access for the file. If no change of date is desired, specify 0. If the statistics function is not selected, these bytes are ignored.
19–20	If the change file backup statistics function is selected in byte 4 (code 4), these bytes specify a new date of creation for the file. If no change of date is desired, specify 0. If the statistics function is not selected, these bytes are ignored.
21–22	If the change file backup statistics function is selected in byte 4 (code 4), these bytes specify a new time of creation for the file. If no change of time is desired, specify 0. If the statistics function is not selected, these bytes are ignored.
23–24+	If N% in byte 3 is 0, specify the name of the device that contains the file you wish to modify. The device must be a disk and a specification of 0 in bytes 23 and 24 indicates the public disk structure. If N% is non-zero, these bytes are ignored.
25–26+	If N% in byte 3 is 0, specify the unit number and unit number flag associated with the file you wish to modify. If N% is non-zero, these bytes are ignored.
27–30	Not used.

Data Returned:

Byte(s)	Meaning
1	Not used.
2	The file characteristics, as follows: byte 2=2 file is placed. byte 2=4 file will be cached sequentially, if at all. byte 2=16 file is contiguous. byte 2=128 file will be cached when open.
3–4	If the file's VBN was passed in byte 16 and file retrieval information (code 16) was requested in byte 4 (see Data Passed), these bytes contain the DCN of the file's VBN. Note that these bytes return 0 if the specified VBN is larger than the file size or if the file was not placed and function code 2 was not passed in byte 4.
5–26	File attribute data; unused words are filled with zeroes.
27–30	The file's run-time system name in Radix-50 format.

Possible Errors:

	Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT	The file or account specified in bytes 5 through 12 is not present on the disk.	5
?I/O CHANNEL NOT OPEN	The channel specified in byte 3 is not open.	9
?PROTECTION VIOLATION	The file open on the channel specified in byte 3 is not a disk file or the job lacks the privilege required for the specified operation.	10
?ILLEGAL SYS( ) USAGE	The file open on the specified channel is not a disk file or is a user file directory.	18

Discussion: This call supplements the functions of the change file run-time system name call (SYS call -18, Section 7.2.19) and the change file statistics call (SYS call -11, Section 7.2.9) to support large files and to add support for file placement. You can also use this call to obtain a file's run-time system name and attribute data without opening the file.

The run-time system name field (see Data Passed, bytes 4 and 27 through 30) in the accounting entry of the file's User File Directory contains file size information for large files. The two-word run-time system name field is decoded as follows:

1. If the first word is non-zero, the data in both words is the run-time system name. The file size is limited to 65535 blocks.
2. If the first word is 0, the low order byte of the second word contains the most significant bits of the file size. The file size is limited to 2**23-1 blocks. The high order byte of the second word is reserved and must be 0.

The following restrictions apply to large files:

1. Because a runnable file cannot have both a run-time system name and a most significant bit indication in the field, large files are not runnable.
2. You cannot extend a compiled file beyond block 65535. An attempt to extend a compiled file past block 65535 results in a ?PROTECTION VIOLATION error (ERR=10).
3. You cannot rename a file that is larger than 65535 blocks with the intent of assigning a compiled protection code. The attempt is rejected with no error and the compiled bit remains off.

4. When you extend a file past block 65535, it loses its run-time system name.

5. You cannot change the run-time system name of a file that is larger than 65535 blocks. The attempt results in a ?PROTECTION VIOLATION error.

6. You cannot change the run-time system name of a compiled file to two words of zeroes. The attempt results in a ?PROTECTION VIOLATION error. Note that you can perform this operation on a non-compiled file.

7. You cannot change the run-time system name of any file to a zero word followed by a non-zero word.

To place a file in a particular position of the disk, you specify the desired disk DCN (Device Cluster Number) as returned in bytes 3 and 4 of this call in the file specification /POSITION switch (see the *RSTS/E System User's Guide*). The Monitor attempts to place the first block of the file at or after the specified DCN. If the file placement is successful, the placed bit (bit 1, mask value 2) in the file's UFD entry is set (see SYS calls -10 and -23, Section 7.2.4.1). If the file placement is not successful, the first block of the file is placed at the lowest free block on the disk, the UFD placed bit is not set, and no error is returned.

7.2.25 Return Job Status — Privileged and Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(26%), the return job status code.
3	CHR$(J%), where J% is the number of the job for which status is desired. If J% is 0%, information on the caller's job is returned. If the caller is not privileged, J% is forced to 0%.
4	CHR$(S%), where S% is 0% or 1%. The value of S% determines the information returned on the job (see Data Returned).
5–30	Not used.

Data Returned:

If S% is 0%:

Byte(s)	Meaning
1	The calling job's job number times two.
2	Not used.
3	Job number times two of the job for which data is being returned. If the caller is not privileged, this byte is equal to byte 1.
4	Keyboard number of the job's console. If the number is negative, the job is detached and the number is the one's complement of the keyboard number.
5–6	If the job is attached to a pseudo keyboard, byte 5 contains the controlling job's job number times two plus one; otherwise byte 5 is 0. If the job is swapped out, byte 6 contains the job's swap slot location; otherwise byte 6 is 0.
7–8	The job's logged in CPU time (least significant word) for the current session in tenths of a second.
9–10	The job's current connect time in minutes.
11–12	The job's current KCTs (least significant word) for this session.
13–14	The job's accumulated device time for the current session in minutes.
15	The most significant byte of the job's KCT.
16	The most significant byte of the job's CPU time.

17–20	The job's name in two Radix–50 words.
21–22	The job's project-programmer number.
23–26	The name of the job's default run-time system in two Radix–50 words.
27–30	The name of the job's current run-time system in two Radix–50 words.

If S% is 1%:

Byte(s)	Meaning
1	The calling job's job number times two.
2	Not used.
3	Job number times two of the job for which data is being returned. If the caller is not privileged, this byte is equal to byte 1.
4	Keyboard number of the job's console. If the number is negative, the job is detached and the number is the one's complement of the keyboard number.
5–6	The job's current flag word.
7	The job's current IOSTS byte.
8	The job's current information posting byte.
9–10	The job's current JBSTAT word.
11–12	The job's current JBWAIT word.
13	The size of the job's current user memory area in K words.
14	The job's control word from its memory control sub-block.
15–16	The job's current physical address in 32 word increments.
17	The job's priority.
18	The job's alloted run burst in tenths of a second.
19	The job's maximum allowable memory size in K words.
20	The value at offset 6 in the job's work block. This value is usually the channel number (times two) on which the job is performing an I/O operation.

21–22	If bytes 9 through 12 indicate that the job is in a keyboard wait state, bytes 21 and 22 contain the value at offset 12 (octal) in the job's work block. This value is the timeout parameter for input from the terminal. If the value is negative, it implies that the terminal is in a keyboard monitor (CTRL/C) input wait state. If bytes 9 through 12 indicate that the job is in an I/O stall for a non-keyboard device, bytes 21 and 22 contain the generic name (in ASCII) of the device for which the job is stalled. If bytes 9 through 12 indicate that the job is in a FIP wait state, byte 21 contains the byte value corresponding to the currently executing FIP function and byte 22 has no meaning.
23–24	The value at offset 16 (octal) in the job's work block. This value is usually an internal code that specifies whether the job is reading or writing on the current I/O channel. If the job is in an I/O wait state and this value is 2, the I/O operation is a read; if this value is 4, the I/O operation is a write.
25–26	A pointer to the beginning of the job's Job Data Block.
27–28	A pointer to the beginning of the job's second Job Data Block.
29–30	If the job is a receiver, these bytes contain the job's receiver identification block; otherwise, these bytes are 0.

Possible Errors:

Meaning	ERR Value
?PROTECTION VIOLATION The calling job is not logged in.	10
?ILLEGAL SYS() USAGE The calling job number is less than zero or greater than JOBMAX.	18

7.2.26 Spooling — Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-28%), the spool request code.
3–4	Not used.
5–6+	The project-programmer number of the file to spool. If bytes 5–6 are zero, the current user account is used. Wildcards are not allowed.
7–10+	The filename (which can include wildcards), in Radix–50 format, of the file to spool.
11–12+	The file extension (which can include wildcards), in Radix–50 format, of the file to spool.
13–14	The two character ASCII device name to which the file is spooled. If bytes 13–14 are zero, LP is used.
15	The unit number of the spooling device specified in bytes 13–14.
16	The unit number flag of the spooling device specified in bytes 13–14 (–1 if byte 15 contains an actual unit number, 0 if bytes 13–14 contain a generic device name).
17–18	Reserved, must be zero.
19–20	The flag word passed to QUEMAN (these values can be combined), as follows:
	1% file is spooled with FORTRAN carriage control; equivalent to QUE /TYP:FTN option.
	2% restart; equivalent to QUE /RE option.
	4% delete the file after spooling; equivalent to QUE /DE option.
	8% binary file; equivalent to QUE /BI option.
	16% end; equivalent to QUE /END option.
	32% no header; equivalent to QUE /NH option.
21–22	Not used.
23–24+	The device name where the file to be spooled can be found. The device must be a disk. If bytes 23–24 are zero, SY (the public structure) is used.
25+	The unit number of the device containing the file to be spooled.
26+	The 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 The number of messages pending for the queue is at its declared maximum. This may be a transient condition; retry the operation.	4
?CAN'T FIND FILE OR ACCOUNT The account specified in bytes 5–6 does not exist on the device specified, the filename or extension specified in bytes 7–12 cannot be found, or QUEMAN is not installed as a message receiver.	5
?NOT A VALID DEVICE An attempt was made to spool a file to a spooling device that had a unit number greater than 7 or the file to be spooled is contained on an invalid device.	6
?PROTECTION VIOLATION An attempt was made to queue a file to which the user did not have read access, queue a compiled file, or queue a file which has the privileged <128> bit set.	10
?DEVICE HUNG OR WRITE LOCKED This error is caused by a hardware condition. For example, the specified disk could not be accessed.	14
?DISK PACK IS NOT MOUNTED The specified disk device is not mounted; logically mount the disk with UTILTY or UMOUNT.	21
?DISK PACK IS LOCKED OUT The disk is in a locked state. Execute the call under a privileged account to override this condition.	22
?DEVICE NOT FILE STRUCTURED The device specified in bytes 23–24 of the call is not a file-structured device.	30
?NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. This may be a transient condition; retry the operation.	32

Discussion: This call allows you to spool one or more files without chaining to QUE.

When this call is executed, the Monitor performs the following checks:

1. ensures that the specified filename is legally formatted.
2. ensures that the specified device (the device containing the specified file) is a mounted RSTS/E disk and that the user has write access to it.
3. if there are no wildcards specified in bytes 7–12, ensures that the specified file exists and that the user has read access to it.
4. performs all appropriate send/receive buffer quota checks and ensures that QUEMAN is available (not hibernating).

If any of these conditions are not met, the call is aborted and an error is returned (see Possible Errors).

7.2.27 Snap Shot Dump — Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-27%), the snap shot dump code.
3-30	Not used.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT The call attempted to write data to the file CRASH.SYS but crash dump was not enabled. To enable crash dump, the system manager must answer yes to the CRASH DUMP question during system initialization.	5
?ILLEGAL SYS USAGE A non-privileged user attempted to execute this call.	18
VARIOUS DEVICE DEPENDENT ERRORS The call also returns device dependent errors such as ?DEVICE HUNG OR WRITE LOCKED and ?DISK PACK IS NOT MOUNTED.	

Discussion: The execution of this call writes the current monitor image executing in memory to the crash dump file [0,1] CRASH.SYS. The contents of the CRASH.SYS file can be analyzed with the ANALYS program as described in the *RSTS/E System Manager's Guide.*

7.2.28 Date and Time Conversion — Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(20%), the date and time conversion code.
3–4	CHR$(D%)+CHR$(SWAP%(D%)), where D% is the date to be converted or 0% for the current date.
5–6	CHR$(D%)+CHR$(SWAP%(D%)), where: D%=0 use the system default format. D%<0 use alphabetic date format. D%>0 use ISO numeric date format.
7–16	Not used.
17–18	CHR$(T%)+CHR$(SWAP%(T%)), where T% is the time to be converted or 0% for the current time.
19–20	CHR$(T%)+CHR$(SWAP%(T%)), where: T%=0 use the system default format. T%<0 use AM/PM time format. T%>0 use 24-hour time format.
21–30	Not used.

Data Returned:

Byte(s)	Meaning
1	The current job number times two.
2	Not used.
3–6	Same as data passed.
7–16	The date string, padded to the right with zeroes (NUL characters).
17–20	Same as data passed.
21–30	The time string, padded to the right with zeroes (NUL characters).

Possible Errors: No errors are possible; however, if bytes 3–4 or 17–18 contain illegal date or time values, random output is generated.

Discussion: This call can be used in programs that need to override the system date and time defaults.

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.* 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
?ODD ADDRESS TRAP The address specified as an argument to PEEK is odd or an attempt is made to reference a non-existent or odd address.	33
?MEMORY MANAGEMENT VIOLATION The address specified as an argument to PEEK is illegal (not mapped in the monitor).	35

7.3.1 Fixed Locations in Monitor

The 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.

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

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). For example: J% = (PEEK(518%) AND 255%)/2% where J% is the user's 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, available to both privileged and non-privileged users, is to execute the return job status SYS call (F0=26) described in Section 7.2.25.

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. The FIP SYS call number 9 returns an even number (KBn:=x/2) if the job is attached and an odd number (KBn:=(NOT x)/2) if detached. However, this method will not work if the job was created by another user.

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.* 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).** 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/E information.

* 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/E Network Programming in BASIC-PLUS and BASIC-PLUS-2* manual.

** The receiving job must be privileged to issue a Declare Receiver system call.

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%)+...+O$)
```

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$(6%)	is the system function code for a call to the file processor (that is, the FIP call).
+	is the concatenation operator required between function, subfunction, and argument codes.
CHR$(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 zeroes 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. Also note that because the Send/Receive call uses CPU resources, excessive use of the call can degrade system performance.

8.3 Declare Receiver - Privileged

Data Passed:

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(22%), the send/receive function code.
3	CHR$(1%), the declare receiver subfunction code.
4	CHR$(0%), reserved - must be zero.
5-10	Name, the receiver's logical name. The receiver's name is a 1- to 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$(O%), the object type code. Object type codes are defined in the *RSTS/E DECnet/E Network Programming in BASIC-PLUS and BASIC-PLUS-2* manual. Legal values are 0 through 127. For local interjob communication, the object type code is zero unless DECnet/E is being used.
22	CHR$(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 Non-privileged. 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/E Network Programming in BASIC-PLUS and BASIC-PLUS-2* manual describes network links. For local interjob communication, this bit should be zero.

S% Network Single Links/No Single Links.

This bit controls the queuing of single network links. For more information, see the *RSTS/E DECnet/E Network Programming in BASIC-PLUS and BASIC-PLUS-2* manual. For local interjob communication, this bit should be zero.

23–24 B%, the buffer maximum in the form CHR$(B%)+CHR$(SWAP%(B%)).

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%), the message maximum.

The message maximum can be any value between one and 255. See the Discussion below.

26 CHR$(L%), the 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 is returned.

Possible Errors:

Meaning	ERR Value
?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.	3
?PROTECTION VIOLATION The caller was non-privileged at the time of the receiver declaration.	10

?NAME OR ACCOUNT NOW EXISTS 16

The logical name passed in Bytes 5–10 is being used by another job.

?ILLEGAL SYS() USAGE 18

An inconsistency exists in the arguments passed.

?NO BUFFER SPACE AVAILABLE 32

When the job attempted to declare itself as a receiving job, there were no small buffers available for the declaration arguments. Because small buffers are dynamically allocated, wait and then retry the operation.

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 – Privileged and Not Privileged

Data Passed:

Bytes	**Meaning**
1	CHR$(6%),the SYS call to FIP.
2	CHR$(22%),the send/receive function code.
3	CHR$(-1%),the send local data message subfunction code.
4	CHR$(J%),the 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.
5–10	N$, the receiver's logical name. The receiver's name is a 1-to 6-character, left-justified, ASCII string, that is padded to six characters with spaces. This field is used only if Byte 4 is CHR$(0%).
11	CHR$(C%), the channel number for the I/O buffer that contains the data portion of the message. If this byte is zero, a string beginning at Byte 41 contains the data portion of this message (if any). If this byte contains a channel number (any value from 1 to 12), a buffer defined by the length and offset values contains the data portion of this message. The message data (up to 512 bytes) should be left-justified in the buffer for channel C% beginning at the offset value defined in Bytes 15-16. Channel 0 can be used for the I/O buffer if 128 is added to the channel number; that is, CHR$(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%, the length (in bytes) of the message to send from the channel buffer in the form CHR$(L%)+CHR$(SWAP%(L%)). 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%, the offset value in the form CHR$(O%)+CHR$(SWAP%(O%)). 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$(0%), reserved - must be zero.
21-40	P$, the 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$, the 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 is returned.

Possible Errors:

Meaning	ERR Value
?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 or the number of pending messages is too small. This error can also occur if the message receiver is hibernating. Because the hibernating receiver is unable to process messages, the system sets XOFF for messages sent to it and thus minimizes the number of small buffers that would be tied up.	4
?CANT 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.	5
?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.	9

?PROTECTION VIOLATION Some access violation has occurred. Either the sender is non-privileged and the receiver requires senders to be privileged, or the receiver does not allow any local senders.	10
?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.	18
?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.	31
?NO BUFFER SPACE AVAILABLE System buffers are not currently available to store this message. A later retry may proceed without error.	32

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 non-zero 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. Because it does not require a receiver table search, 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 non-privileged 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 - Privileged and Not Privileged

Data Passed:

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(22%), the send/receive function code.
3	CHR$(2%), the receive subfunction code.
4	CHR$(S%+T%+L%+N%), the 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.

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%), the sender selection.

This byte is ignored if both L%=0% and N%=0%.

Any non-zero 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/E Network Programming in BASIC-PLUS and BASIC-PLUS-2* manual for further details.

6 CHR$(Q%), the 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 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/E Network Programming in BASIC-PLUS and BASIC-PLUS-2* manual 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%), the channel number for the I/O buffer to receive messages.

If C% is between 1 and 12, the system returns 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%, the maximum message length (in bytes) desired on this receive in the form CHR$(L%)+CHR$(SWAP%(L%)).

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, ERR=4 (?NO ROOM FOR USER ON DEVICE) occurs. 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%, the offset from the start of the buffer in the form CHR$(O%)+CHR$(SWAP%(O%)). 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$(0%) reserved - must be zero.
21–26	Ignored.
27–28	CHR$(T%), the sleep time in seconds in the form CHR$(T%)+CHR$(SWAP%(T%)). 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=5) is returned.
29–40	CHR$(0%) reserved - must be zero.

Data Returned: Local Data Message

Bytes	Meaning
1–2	Not meaningful - should be ignored.
3	CHR$(-1%), the local data message subfunction code.
4	CHR$(J%), the 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 of the sender.
7–8	Not meaningful - should be ignored.
9–10	R%, the number of bytes remaining in the data portion of the message. 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%, the 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 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$, the user 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:

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE The maximum message length desired on the receive is declared as zero-length. Change the receive declaration to include a non-zero value for message length.	4
?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, or that the sleep timer has expired. 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).	5
?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.	9
?ILLEGAL SYS() USAGE The job is not a declared receiver. Before any receive can succeed, the job must be entered in the receiver list.	18
?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.	31

?NO BUFFER SPACE AVAILABLE When the job attempted to declare itself as a receiving job, there were no small buffers available for the declaration arguments. Because small buffers are dynamically allocated, wait and then retry the operation.	32

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 or 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 four events:

1. A local or DECnet message is queued for the job.
2. The sleep timer expires.
3. A delimiter is typed on any of the terminals owned by the job.
4. The number of logins is set to one (that is, logins are disabled).

In all cases, the job is awakened with an error (ERR=5) but is not passed a message. To obtain a pending message, the job must execute the receive call again. Because 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 messages 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%	non-zero	Selects job 0; used by error logging programs to select messages from monitor routines.
		non-zero	–	Selects local message by job number times 2 in byte 5.
8%	0%	0%	0%	Selects first Network message.
		0%	non-zero	Selects first Network single message only; no link exists (uses Link Address=0).
		non-zero	–	Selects Network message by link (user Link Address) in byte 5.

(F0= 22)

8.6 Remove Receiver – Privileged and Not Privileged

Data Passed:

Byte(s)	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(22%), the send/receive function code.
3	CHR$(0%), the remove subfunction code.
4	CHR$(J%), the job number (times 2) of the job to remove, or CHR$(0%) to remove the calling job. If J%=0%, the calling job need not be privileged. If J% is non-zero, the caller must be privileged.
5–40	CHR$(0%) reserved - must be zero.

Data Returned: No meaningful data is returned.

Possible Errors:

Meaning	ERR Value
?PROTECTION VIOLATION The caller is non-privileged and has attempted to remove another job (i.e., Byte 4 is non-zero).	10
?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.	18

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 receiver 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=5), 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 because the program does not 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 * 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 because the program preserves 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
```

The sender's job number and the number of bytes in the next pending message are known. 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%(S%(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%)
```

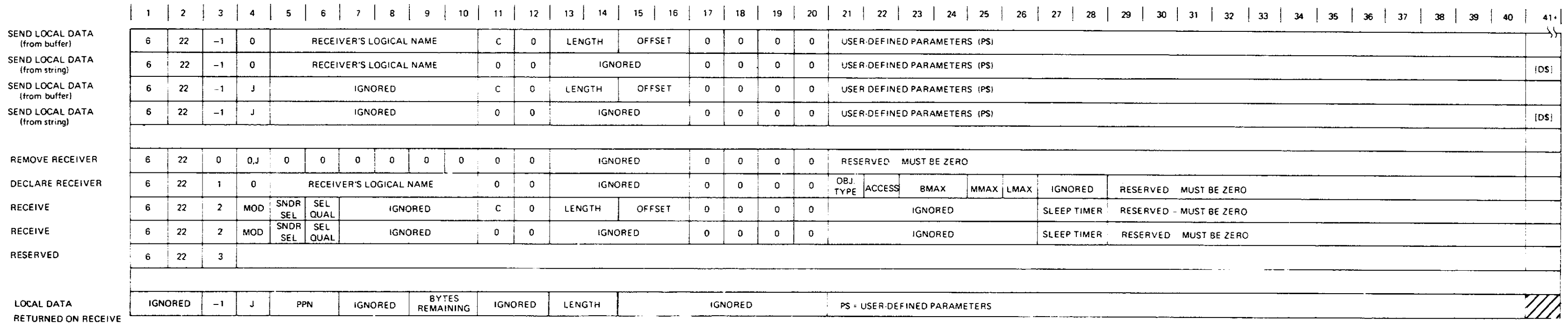
The example concludes with a remove receiver call and a close of the channel buffer used to receive messages.

```
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.

Figure 8-1: Summary of Send/Receive Data



	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 (PS)																				
SEND LOCAL DATA (from string)	6	22	−1	0	RECEIVER'S LOGICAL NAME						0	0	IGNORED				0	0	0	0	USER-DEFINED PARAMETERS (PS)																				[D$]
SEND LOCAL DATA (from buffer)	6	22	−1	J	IGNORED						C	0	LENGTH		OFFSET		0	0	0	0	USER DEFINED PARAMETERS (PS)																				
SEND LOCAL DATA (from string)	6	22	−1	J	IGNORED						0	0	IGNORED				0	0	0	0	USER-DEFINED PARAMETERS (PS)																				[D$]
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						PS = USER-DEFINED PARAMETERS																				▨

F-MK-00038-00

Chapter 9
Programming Conventions

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 SYS typed at a logged out terminal, LOGIN chains to the SYSTAT program in the system library account. The following statement in LOGIN ensures that action.

```
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 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 the LOGIN source 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 statement 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(CCL) 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 that all CCL commands be defined at system start up time. At the start of time sharing, INIT 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 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.

Note that because of the manner in which the parser interprets CCL commands, the system manager should ensure that similar commands are defined in the correct order. That is, MACRO must be defined before MAC during system start up.

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 (ERR=67). 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.

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).

If BASIC-PLUS finds that the line number is non-zero, 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 permanent privilege and the /DET switch was specified.

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 may not perform dynamic memory expansion during the execution of a program, as BASIC-PLUS does, and may 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 31K words for all users. This action means that all users run with a swap maximum of 31K words.

To modify the LOGIN source program to set a swap maximum less than 31K words for non-privileged accounts, alter the J% = 31% 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% = 31%
         \ J% = 31% 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 31% in the statement J% = 31% 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 31K words for privileged accounts, change the J% = 31% expression in the second physical line at line number 15010. The statement on this line checks for a 1 as the project number of the account.

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

```
COMPILE SY0:LOGIN$
READY
NAME "LOGIN.BAC$" AS "LOGIN.BAC$<232>"
READY
```

It is recommended that the system manager not replace the original LOGIN source file with the modified version.

Chapter 10
DMC11 Interprocessor Link

10.1 Using the DMC11 Interprocessor Link in Point-to-Point Configurations

The DMC11 Network Link (device XM: on RSTS/E) provides high speed local or remote interconnection of computers over a serial synchronous link. It uses the Digital Data Communications Protocol (DDCMP) to provide data transmission and uses Non-Processor Request (NPR) data transfers to and from memory to provide high throughput and minimize processor overhead.

Normally, the DMC11 is used by the DECnet/E Network Service Protocol (NSP) package, which supports multiple node networks, user data security, multiple logical links over a single physical link, and other network features. When used by NSP, the DMC11 is not available to the RSTS/E user except via NSP. However, in point-to-point configurations, DECnet/E may not be needed. In these cases, the user may access the DMC11 directly from his program to obtain a communication link with another processor. NSP need not even be configured into the RSTS/E system.

10.1.1 The OPEN Statement

The DMC11 is not a file structured device. However, certain parameters must be specified at open time to establish the device's operating mode. Also, when the OPEN statement is executed on a DMC with an autoanswer/autodial phone connection, the DTR (Data Terminal Ready) is automatically raised to enable data transmission.

10.1.1.1 MODE Value — The MODE value used when opening a DMC11 indicates whether the unit is to be run in full duplex, as a half duplex primary station, or as a half duplex secondary station. These states are described in the *PDP-11 Peripherals Handbook*. To cause the DMC11 to hang up a phone connection when it receives a DDCMP restart, add 256 to the specified MODE value. To specify full duplex, omit the MODE in the OPEN statement, or specify a MODE value of zero. To specify half duplex primary station, use a MODE value of 1024. To specify half duplex secondary station, use a mode value of 3072.

10.1.1.2 CLUSTERSIZE Value — To ensure that messages from the remote processor to the local RSTS/E system are received without need for retransmission, the DMC11 allocates one or more receive buffers to the unit when it is opened. Whenever a message is received over the link, it is placed in one of the allocated buffers. That buffer is then placed on a queue of received messages. When the user issues a GET statement on the open channel of the DMC11, the message is copied from the system buffer to the user's buffer, and the system buffer is released.

Since a buffer on the receive complete queue is no longer available for use by the DMC11, the driver tries to replace it with another buffer from the Monitor's extended buffer pool or from the small buffer pool. The number of buffers which the driver attempts to keep allocated to the DMC11 receiver side is called the buffer quota for the unit, and is specified at OPEN time as the CLUSTERSIZE value. Any number from 1 to 127 is valid as a buffer quota, but values above seven are not recommended except when very high traffic on a one-megabaud line is expected; allocating too many buffers to the DMC11 needlessly ties up system resources. However, if the buffer quota is too low, overrun errors will occur on the unit.

10.1.1.3 FILESIZE Value — The value used in the FILESIZE option at open time specifies the size of the buffers allocated to the DMC11 receiver. This value limits the length of a received message, and must be between 1 and 632 inclusive. If the remote processor sends a message larger than the receiver buffer size, the message will be lost and the DMC11 will halt operation. Note that the 632 byte limit on receive buffer size does not limit the length of transmitted messages. The DMC driver limits transmitted message lengths to a maximum of 8000 bytes. However, the user must be careful to stay within the remote system's receive buffer size. For example, if the remote system is also RSTS/E, the length of transmitted messages is limited to 632 bytes maximum or a smaller value that is equal to the receive buffer size as established by the FILESIZE value (specified in the OPEN statement for the remote DMC11).

10.1.1.4 Errors — Only two errors specific to the DMC11 can occur at OPEN time. ERR 14 (?DEVICE HUNG OR WRITE LOCKED) occurs if the driver cannot obtain the unit's port in order to initialize it. ERR 32 (?NO BUFFER SPACE AVAILABLE) occurs if the driver cannot obtain a small buffer to use as a DDB extension and a 264 byte buffer to use as the hardware core table.

10.1.2 The GET Statement and RECORD Options

The GET statement copies the next message from the DMC11's queue of received messages into the user's I/O buffer. If the received message is longer than the buffer, the Monitor truncates it with no warning. Therefore, it is good practice to specify a RECORDSIZE in the OPEN statement which is greater than or equal to the FILESIZE value (see Section 10.1.1.3).

The value in the RECORD option of GET statements determines how the program treats message unavailability. If no message is available and the DMC11 is still running (i.e., the physical link is intact), the program may elect to get an error indication immediately or to sleep until a message is received. A RECORD value of 0 (or omitting the RECORD option) tells the Monitor to generate ERR 5 (?CAN'T FIND FILE OR ACCOUNT) immediately. A RECORD value of 8192% tells the Monitor to stall the job until a message is available from the remote processor (in which case the message is returned in the user's buffer as usual) or until a DMC11 error occurs (in which case the program receives ERR 14 (?DEVICE HUNG OR WRITE LOCKED). A RECORD value of 16384%+n%, where n% is a number between 0 and 255, causes the Monitor to put the job to sleep. It will be awakened by any of the following conditions:

1. a message is received on the DMC11.
2. an error occurs on the DMC11.
3. a message is received via the local send/receive mechanism.
4. a delimiter is typed on one of the job's keyboards.
5. the number of logins is set to 1.
6. n seconds have expired and n is not 0.

If the job is awakened because a message is received, the Monitor copies the message to its buffer, just as if the GET had succeeded without sleeping. If it is awakened because an error occurred, it receives ERR 14 (?DEVICE HUNG OR WRITE LOCKED). If it is awakened for any other reason, it receives ERR 5 (?CAN'T FIND FILE OR ACCOUNT).

When the DMC11 driver detects a failure in the physical link, it generates ERR 14 (?DEVICE HUNG OR WRITE LOCKED) and shuts down the unit. Any messages received prior to the hardware failure are returned to the job as it executes GET statements. No error indication appears until the receive complete queue is empty. At that point, the job receives ERR 14 (?DEVICE HUNG OR WRITE LOCKED). The only recourse is to CLOSE the channel on which the unit is open.

10.1.2.1 Count and Status Information — If the 4096% bit is on in the RECORD value in a GET statement, the driver does not return a message from the DMC11. Instead, it returns count and status information to the user. 26 bytes of information are returned in the following format:

Byte(s)	Meaning
1	Number of transmit buffers actually being processed by the DMC11 hardware (0 to 7).
2	Total number of transmit buffers waiting to be sent, including those given to the hardware (i.e., number of uncompleted PUT statements).
3	Number of messages on the receive queue waiting to be given to the job.
4	Unused.
5	Number of receive buffers actually given to the DMC11 hardware.
6	Total number of buffers allocated to the DMC11 receiver, including those given to the hardware.
7–8	Length of the first message on the receive queue (0 if byte 3 is 0).
9	If the DMC11 is not running (see byte 10), this is a code indicating the type of error: 0 Hardware error (see control-out information in bytes 19–20). 1 Unknown control-out operation. 2 Illegal input interrupt. 3 Illegal output interrupt. 4 Unsolicited input interrupt. 5 Unexpected output interrupt. 6 DDCMP maintenance mode. 7 Lost data error. 8 Disconnect code. 9 DDCMP start received. 10 Unibus address timeout on DMC access. 11 Procedure error. 255 Timeout error.
10	Status flags, encoded as a combination of bits: 4 The first transmit since the DMC11 was OPENed is complete, indicating that a link has been established and that further transmits should be timed out. 64 The driver is waiting for buffers to satisfy receive buffer quota. 128 Unit is running. If this bit is off, the DMC11 was halted for the reason given in byte 9.
11–12	Receive buffer size (from the FILESIZE value when the DMC11 was OPENed).
13–14	Operational mode (from the MODE value when the DMC11 was opened).
15–16	Reserved.
17	Receive buffer quota (from the CLUSTERSIZE value when the DMC11 was opened).

18	Reserved.
19–20	Value of SEL6 hardware register at most recent control out interrupt. If the DMC11 was halted due to a hardware error, the specific error (or errors) can be found here. Refer to the description of the DMC11 in the *PDP–11 Peripherals Handbook* for the format of this word.
21–22	Data check count. A data check error occurs when the DMC11 has tried seven retransmissions of a message without success. This indicates that the physical channel is defective or that the remote processor does not have a buffer to receive the message. The DMC11 continues to retry the transmission, and reports a data check error every seven retries. The total number of data check errors which have occurred since the DMC11 unit was opened is returned in this word.
23–24	Timeout count. A timeout error occurs when the DMC11 has received no response from the remote end of the link for 21 seconds. This indicates a broken communications channel or a failure at the other end of the link. The number of timeout errors since the OPEN is returned in this word.
25–26	Overrun count. An overrun error indicates that a message was received but no buffer was available. This is non-fatal, since the remote system will retransmit the message (and possibly log data check errors). Overrun errors can be reduced by increasing the buffer quota for the unit (see CLUSTERSIZE in OPEN). Overrun errors caused by the driver not being able to obtain a buffer allowed by the buffer quota value can be reduced by increasing the size of the XBUF area at the start of the next time-sharing session. The number of overrun errors since the OPEN is returned in this word.

Three errors (data check, timeout, and overrun) which are detected by the DMC11 are only warnings. These are non-fatal, do not cause the unit to halt and the user program is not informed when they occur. However, if any of them occurs frequently, it indicates that the program has set the wrong values for CLUSTERSIZE or FILESIZE, or that there is trouble on the physical line between the two processors. The driver counts the number of times each error occurs, and returns those counts as part of the status information.

The user program never stalls when it issues a count and status request. Furthermore, this request is legal whether or not the unit is running. Thus, it can be used to determine the specific DMC11 problem after the program receives an ERR 14.

10.1.3 The PUT Statement

The PUT statement copies data from the user's I/O buffer to a system buffer and queues the buffer for transmission. The number of bytes to transmit is specified in the COUNT option, and may be from 1 to 8000. (A COUNT value outside that range will generate ERR 31, ?ILLEGAL BYTE COUNT FOR I/O.) If the monitor cannot obtain a buffer big enough to hold the message, it returns ERR 32 (?NO BUFFER SPACE AVAILABLE). The program can sleep for a while and retry the PUT, waiting for adequate buffer space to become available. Note that on a given configuration it may be impossible to obtain a buffer of the proper size. It is good practice to limit the retry operations to a small number after receiving ERR 32.

If the physical link has gone down, the driver will immediately return ERR 14 (?DEVICE HUNG OR WRITE LOCKED). As with the GET statement, the only recourse is to close the channel.

The PUT statement queues messages to the DMC11 to be sent as soon as possible. The user program is not normally notified when the actual message transmission is done, nor whether it is ever done (in case of a physical link failure). This action can be modified by using the RECORD clause of the PUT statement. A RECORD value of 0 (or omitting the RECORD option) tells the Monitor to queue the data for transmision, and the program immediately continues processing. RECORD 8192% tells the Monitor to stall the job until all pending transmissions have completed successfully (in which case the program continues processing normally) or until a DMC11 error occurs (in which case the program receives error 14). RECORD 16384%+n%, where n% is a number between 0 and 255, causes the Monitor to put the job to sleep. It will be awakened by any of the following conditions:

1. all pending transmissions have completed successfully.
2. an error occurs on the DMC11.
3. a message is received via the local send/receive mechanism.
4. a delimiter is typed on one of the job's keyboards.
5. the number of logins is set to 1.
6. n seconds have expired and n is not 0.

If the job is awakened for reason 2, it receives ERR 14 (?DEVICE HUNG OR WRITE LOCKED). If it is awakened for any other reason, it receives no error and continues processing normally. The number of transmissions still outstanding can be found by doing a GET with RECORD 4096% and examining the value in byte 2.

Adding the value 4096% to any of the above MODE values tells the Monitor not to transmit any data, but to do the wait operation specified.

10.1.4 The CLOSE Statement

If a DMC11 unit is open by a user on more than one channel, no CLOSE except the last has any effect. When the last CLOSE is issued, the unit is halted, any received messages not given to the user are discarded, any messages queued for transmission but not transmitted are discarded, and all buffers are returned to the Monitor. It is normally good practice to issue a PUT statement with RECORD 4096%+16384% to wait for all transmissions to complete before executing a CLOSE statement. The CLOSE call cannot fail. When the CLOSE statement is executed on a DMC with an autoanswer/autodial phone connection, the DTR (Data Terminal Ready) is automatically lowered to disable data transmission.

10.1.5 Hardware Errors

Any fatal error detected by the DMC11 (that is, any error not listed as nonfatal in the count and status description) causes the monitor to shut down the link. The monitor will report error 14 to the user job on all subsequent PUT operations and on any GET operations after all queued messages have been received. (GET operations with RECORD 4096% are always legal, whether or not the unit is running.)

Appendix A
Magnetic Tape Label Formats

RSTS/E supports two magnetic tape file label formats; DOS and ANSI. The following sections discuss DOS and ANSI label formats and describe how RSTS/E handles tapes written with these labels. Note that the ANSI label format discussed in these sections refers to the RSTS/E implementation of the American National Standard X3.27–1978 (magnetic tape labels and file structure for information interchange).

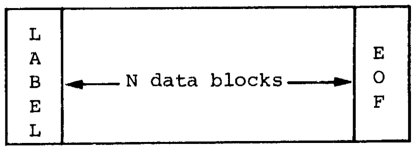
A.1 DOS Magnetic Tape Format

This section describes the labels and data blocks on a magnetic tape 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 magnetic tape under discussion has three files, each containing ten data blocks.

The first part of the magnetic tape is a physical beginning-of-tape (BOT), which is a reflective (metallic) 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 magnetic tape 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 contain zero data blocks, but a label and end-of-file record are always required for each file.

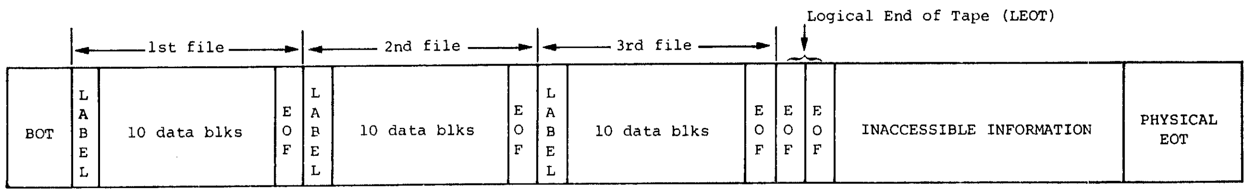
Figure A-1: A DOS Labeled Magnetic Tape File



Q-MK-00039-00

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 magnetic tape has three end-of-file records at this point, signifying a logical end-of-tape (LEOT).

Figure A-2: DOS Magnetic Tape Consisting of 3 Files of 10 Data Blocks Apiece



Q-MK-00040-00

Once the logical end-of-tape is written on the magnetic tape, it can be written over, but it cannot be read over. Therefore, all information beyond the logical end-of-tape is inaccessible.

If a magnetic tape contains no files, three end-of-file records follow the beginning-of-tape marker.

A.1.1 DOS Labels

The record label which specifies the beginning of a magnetic tape 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	Filename	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 magnetic tapes are to be interchanged with DOS–11 systems, a problem may occur since RSTS/E 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 "MTO:ABC[1,1]" FOR OUTPUT AS FILE 1%
```

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 magnetic tape 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 Magnetic Tape Format

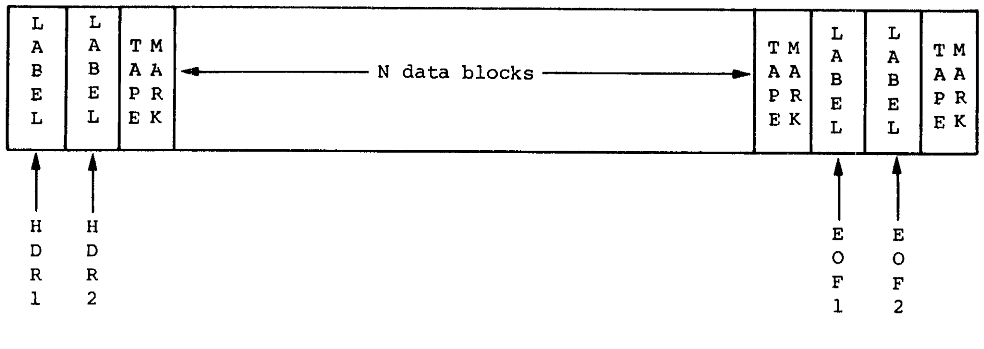
This section describes the labels and data blocks on a single or multi-volume magnetic tape with ANSI labels. Once again, for purposes of explanation, assume that the particular magnetic tape under discussion has three files, each containing ten data blocks.

The first part of the magnetic tape is a physical beginning-of-tape marker. The next item is a volume label (VOL 1). (A volume is simply a reel of magnetic tape. A volume, which may be part of a volume set, may contain part of a file, a complete file, or more than one file.)

The first RSTS/E file actually begins with two labels, called header labels (HDR1 and HDR2). These header labels are followed by an end-of-file or end-of-volume (EOF or EOV) tape mark for single volume or multi-volume, respectively. In this case, ten data blocks are written immediately after the single marker. The data blocks are followed, in order, by another marker, two trailer labels (EOF1 and EOF2 or EOV1 and EOV2), and yet another marker.

All RSTS/E files in ANSI format begin with two header labels. An EOF or EOV tape mark is written on both sides of the data blocks. Two trailer labels follow, and the file is terminated by another EOF or EOV tape mark. When the file is created but no data blocks are written, all the above labels and end-of-file markers are still present. These labels and end-of-file markers are always required for each file.

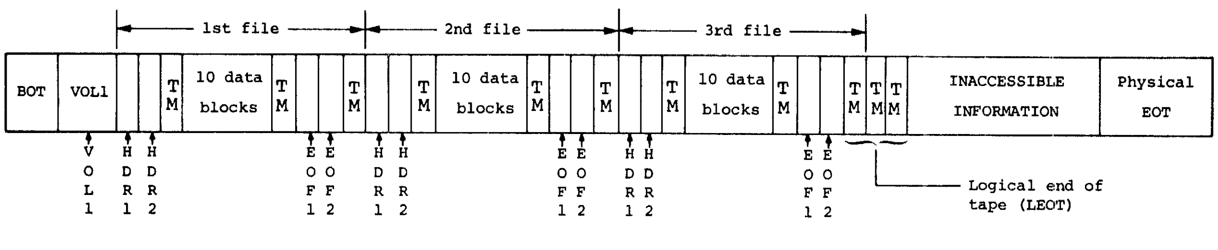
Figure A-3: An ANSI Labeled Magnetic Tape File



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 tape mark written on each side of ten data blocks, two trailer labels and a tape mark.

The third file on the tape is identical to the first and second, and is followed by two tape marks, signifying a logical end-of-tape (LEOT).

Figure A-4: ANSI Magnetic Tape Consisting of 3 Files of 10 Data Blocks Apiece



Once the logical end-of-tape is written on the magnetic tape, it can be written over, but it cannot be read over. Therefore, all information beyond the logical end-of-tape is inaccessible.

A magnetic tape must contain at least one complete set of header and trailer labels. In the case where no file exists on the tape (such as an initialized magnetic tape), a dummy file is present with a complete set of labels and end-of-file tape marks.

A.2.1 ANSI Labels

All ANSI labels written by RSTS/E are 80 bytes long. Each label can be identified by its first three characters: VOL (volume), HDR (header), and EOF or EOV (end-of-file or end-of-volume). 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 magnetic tape 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* D%B4431JJJGGG	The contents of this field are used for volume protection.
52–79	Reserved	Spaces
80	Label Standard Version	3

* 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 byte in the header 1 label.

Table A-3: Header 1 Label Format

Character Position	Field Name and RSTS/E Usage	Contents
1-3	Label Identifier	HDR
4	Label Number	1
5-21	File Identifier (2 to 10 characters FILNAM. or FILNAM.EXT; blank filled)	Any alphanumberic 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. Identifies a section in the file.
32-35	File Sequence Number	Numeric characters; starts at 0001. Identifies a file on the volume.
36-39	Generation Number (0001)	Not supported by RSTS/E; always 0001.
40-41	Generation Version (00)	Not supported by RSTS/E; always 00.
42-47	Creation Date Today's date in specified format	(SPACE)YYDDD or (SPACE)00000 if no date.
48-53	Expiration Date Today's date in specified format	(SPACE)YYDDD or (SPACE)00000 if expired.
54	Accessibility	Space
55-60	Block Count	000000
61-73	System Code DECRSTS/E	Name of system which produced the volume. Padded by spaces.
74-80	Reserved	Spaces

A.2.1.3 Header 2 Label (HDR2) — Table A–4 shows the character position, field name and contents of each byte in the header 2 label.

Table A–4: Header 2 Label Format

Character Position	Field Name and RSTS/E Usage	Contents
1–3	Label Identifier	HDR
4	Label Number	2

5	Record Format
	(U is the default)
	(S is unsupported)
		F = Fixed
		D = Variable
		S = Spanned
		U = Undefined*

6–10	Block Length (512 is the default)	Numeric characters settable by FILESIZE option.
11–15	Record Length	Numeric characters settable by CLUSTERSIZE option.
16–50	System Dependent (M is the default)	Bytes 16–36 (Spaces) Byte 37 = A means first byte of record contains FORTRAN control character. = (Space) means LF character precedes and CR character follows each record. = M means record contains all form control information.
51–52	Buffer Offset(00)	Not supported by RSTS/E; always 00.
53–80	Reserved	Spaces

*U format is not defined by ANSI standard X3.27–1978.

A.2.1.4 End-Of-File or Volume 1 Label (EOF1 or EOV1) — The EOF1 or EOV1 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 label. EOF indicates the end of the file, EOV indicates that the end of the ANSI labeled magnetic tape volume is reached and the current file is continued on another volume. Note that only the RSTS/E utility program PIP reads and writes EOV labels (see the *RSTS/E System User's Guide*).

Table A–5: End-of-File or Volume (EOF or EOV) 1 Record Format

Character Position	Field Name and RSTS/E Usage	Contents
1–3	Label Identifier	EOF or EOV.
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. Identifies a section in the file.
32–35	File Sequence Number	Numeric characters; starts at 0001. Identifies a file on the volume.
36–39	Generation Number (0001)	Not supported by RSTS/E; always 0001.
40–41	Generation Version (00)	Not supported by RSTS/E; always 00.
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	Total filesize in blocks for EOF. Current filesize for EOV.
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 or Volume 2 Label (EOF2 or EOV2) — The EOF2 or EOV2 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 label.

Table A–6: End-of-File or Volume (EOF or EOV) 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 = Undefined*

6–10	Block Length (512 is the default)	Numeric characters settable by FILESIZE option.
11–15	Record Length	Numeric characters settable by CLUSTERSIZE option.
16–50	System Dependent (M is the default)	Bytes 16–36 (Spaces). Byte 37 = A means first byte of record contains FORTRAN control character. = (Space) means LF character precedes and RET character follows each record. = M means record contains all form control information.
51–52	Buffer Offset (00)	Not supported by RSTS/E; always 00.
53–80	Reserved	Spaces

*U format is not defined by ANSI standard X3.27 - 1978.

A.3 Initializing Magnetic Tapes

This section describes how a magnetic tape is initialized (zeroed). See the *RSTS/E System User's Guide* for the procedure for initializing magnetic tapes with PIP.

A magnetic tape written in DOS format is initialized in the following manner:

1. Magnetic tape is rewound.
2. Three end-of-file (EOF) marks are written on the tape.
3. Magnetic tape is again rewound.

A magnetic tape written with ANSI labels is initialized in the following manner:

1. Magnetic tape is rewound.
2. A volume label (VOL1) is written on the tape. The volume identifier is in bytes 5 through 10, in ASCII.
3. Two header labels (HDR1 and HDR2) are written on the tape.
4. Two end-of-file (EOF) tape marks are written on the tape.
5. Two trailer labels (EOF1 and EOF2) are written on the tape.
6. Three end-of-file (EOF) tape marks are written on the tape.
7. Magnetic tape is again rewound.

Notice that in the case of magnetic tapes written with ANSI labels, the two header labels (HDR1 and HDR2), two EOF tape marks, two trailer labels (EOF1 and EOF2) and a final EOF tape mark comprise a dummy file. Also, in both the DOS format and the ANSI format case, three EOF tape marks are the last items written on the tape. These three EOF tape marks form the logical end-of-tape (LEOT).

Appendix B
Card Codes

The RSTS/E card reader driver can be configured for one of four different punched card codes. These are; ANSI codes, DEC029 codes, DEC026 codes and 1401 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.

Table B–1: Card Reader Codes

Character	ASCII$_{10}$	DEC029	DEC026	1401
{	123	12 0	12 0	unused
}	125	11 0	11 0	unused
SPACE	32	NONE	NONE	NONE
!	33	12 8 7	12 8 7	11 0
"	34	8 7	0 8 5	0 8 2
#	35	8 3	0 8 6	8 3
$	36	11 8 3	11 8 3	11 8 3
%	37	0 8 4	0 8 7	0 8 4
&	38	12	11 8 7	12
'	39	8 5	8 6	12 8 4
(	40	12 8 5	0 8 4	8 7
)	41	11 8 5	12 8 4	0 8 7
*	42	11 8 4	11 8 4	11 8 4
+	43	12 8 6	12	0 8 5
,	44	0 8 3	0 8 3	0 8 3
–	45	11	11	11
.	46	12 8 3	12 8 3	12 8 3
/	47	0 1	0 1	0 1

(continued on next page)

Table B-1: Card Reader Codes (Cont.)

Character	ASCII$_{10}$	DEC029	DEC026	1401
0	48	0	0	0
1	49	1	1	1
2	50	2	2	2
3	51	3	3	3
4	52	4	4	4
5	53	5	5	5
6	54	6	6	6
7	55	7	7	7
8	56	8	8	8
9	57	9	9	9
:	58	8 2	11 8 2	8 5
;	59	11 8 6	0 8 2	11 8 6
<	60	12 8 4	12 8 6	12 8 6
=	61	8 6	8 3	11 8 7
>	62	0 8 6	11 8 6	8 6
?	63	0 8 7	12 8 2	12 0
@	64	8 4	8 4	8 4
A	65	12 1	12 1	12 1
B	66	12 2	12 2	12 2
C	67	12 3	12 3	12 3
D	68	12 4	12 4	12 4
E	69	12 5	12 5	12 5
F	70	12 6	12 6	12 6
G	71	12 7	12 7	12 7
H	72	12 8	12 8	12 8
I	73	12 9	12 9	12 9
J	74	11 1	11 1	11 1
K	75	11 2	11 2	11 2
L	76	11 3	11 3	11 3
M	77	11 4	11 4	11 4
N	78	11 5	11 5	11 5
O	79	11 6	11 6	11 6
P	80	11 7	11 7	11 7
Q	81	11 8	11 8	11 8
R	82	11 9	11 9	11 9
S	83	0 2	0 2	0 2
T	84	0 3	0 3	0 3
U	85	0 4	0 4	0 4
V	86	0 5	0 5	0 5
W	87	0 6	0 6	0 6
X	88	0 7	0 7	0 7
Y	89	0 8	0 8	0 8
Z	90	0 9	0 9	0 9
[	91	12 8 2	11 8 5	12 8 5
\	92	0 8 2	8 7	0 8 6
]	93	11 8 2	12 8 5	11 8 5
↑ or ^	94	11 8 7	8 5	unused
← or _	95	0 8 5	8 2	12 8 7

EOF is 12-11-0-1 punch or a 12-11-0-1-6-7-8-9 punch.

Appendix C Error Messages

Messages in RSTS/E are generated for BASIC–PLUS errors* 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.

* 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, BASIC–PLUS-2 and FORTRAN-IV. For these error messages, consult the appropriate User's Guides.

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.

```
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 value. Whenever an error occurs with trapping in effect, the system checks the error variable (ERR), which contains the appropriate decimal error value in the range 0 to 127. An error with a value between 1 and 69 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 value is between 70 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 values. A program can use these error values to differentiate errors. Non-recoverable errors are in alphabetical order without error values because a program can not use these values in an error handling routine.

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.No space or tab is allowed after the ?.
	Information	A message beginning with neither a question mark nor a percent is for information only.

BATCH examines the first character position of each error to determine the severity of the error.

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. Instructions for filling out the SPR are given in Section C.3.

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. Zero 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 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

Message Printed	ERR Value
(SYSTEM INSTALLATION NAME) The error code 0 is associated with the system installation name and is used by system programs to print identification lines.	0
?BAD DIRECTORY FOR DEVICE The directory of the referenced device is in an unreadable format. The magnetic tape 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.	1
?ILLEGAL FILE NAME The specified filename 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 (@).	2

Message Printed	ERR Value
?ACCOUNT OR DEVICE IN USE An attempt to reassign or dismount the device fails 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.	3
?NO ROOM FOR USER ON DEVICE Storage space allowed for the current user on the specified device has been used or the device as a whole is too full to accept further data.	4
?CAN'T FIND FILE OR ACCOUNT The specified file or account number was not found on the specified device. The CCL command to be deleted does not exist.	5
?NOT A VALID DEVICE The supplied device specification 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.	6
?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)	7
?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.	8
?I/O CHANNEL NOT OPEN An attempt is made to perform I/O on one of the twelve channels which has not been previously opened in the program.	9

Message Printed	ERR Value
?PROTECTION VIOLATION The user was prohibited from performing the requested operation because the 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).	10
?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.	11
?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 a hardware condition. Report such occurrences to the system manager. (See the discussion at beginning of Appendix C.)	12
?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.	13
?DEVICE HUNG OR WRITE LOCKED User should check hardware condition of the requested device. Possible causes of this error include a line printer out of paper or high-speed reader being off-line.	14
?KEYBOARD WAIT EXHAUSTED Time requested by WAIT statement has been exhausted with no input received from the specified keyboard.	15
?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.	16
?TOO MANY OPEN FILES ON UNIT Only one open DECtape output file is permitted per DECtape drive. Only one open file per magnetic tape drive is permitted.	17
?ILLEGAL SYS() USAGE Illegal use of the SYS system function.	18

Message Printed	ERR Value
?DISK BLOCK IS INTERLOCKED The requested disk block segment is already in use (locked) by some other user.	19
?PACK IDS DON'T MATCH The identification code for the specified disk pack does not match the identification code already on the pack.	20
?DISK PACK IS NOT MOUNTED No disk pack is mounted on the specified disk drive.	21
?DISK PACK IS LOCKED OUT The specified disk pack is mounted but temporarily disabled.	22
?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.	23
?DISK PACK IS PRIVATE The current user does not have access to the specified private disk pack.	24
?DISK PACK NEEDS "CLEANING" Non-fatal disk mounting error; use the ONLCLN system program.	25
?FATAL DISK PACK MOUNT ERROR Fatal disk mounting error. Disk cannot be successfully mounted.	26
?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.	27
?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.	28
?CORRUPTED FILE STRUCTURE Fatal error in CLEAN operation.	29

Message Printed	ERR Value
?DEVICE NOT FILE STRUCTURED An attempt is made to access a device, other than a disk, DECtape, or magnetic tape 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.	30
?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.	31
?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.	32
?ODD ADDRESS TRAP This error occurs when an attempt is made to reference a nonexistent address, reference an odd address using a word instruction, or perform a PEEK function with an odd or nonexistent parameter.	33
?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 C.)	34
?MEMORY MANAGEMENT VIOLATION This hardware error occurs when an illegal Monitor address is specified using the PEEK function.	35
?SP STACK OVERFLOW An attempt to extend the hardware stack beyond its legal size is encountered. (See discussion at beginning of Appendix C.) (SPR)	36

Message Printed	ERR Value
?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 C.)	37
?MEMORY PARITY FAILURE A parity error was detected in the memory occupied by this job. (See discussion at beginning of Appendix C.)	38
?MAGTAPE SELECT ERROR When access to a magnetic tape drive was attempted, the selected unit was found to be off line.	39
?MAGTAPE RECORD LENGTH ERROR When performing input from magnetic tape, the record on tape was found to be longer than the buffer designated to handle the record.	40
?NON-RES RUN-TIME SYSTEM The referenced run-time system has not been loaded into memory and is therefore non-resident.	41
?VIRTUAL BUFFER TOO LARGE Virtual core buffers must be 512 bytes long.	42
?VIRTUAL ARRAY NOT ON DISK A non-disk device is open on the channel upon which the virtual array is referenced.	43
?MATRIX OR ARRAY TOO BIG In-core array size is too large.	44
?VIRTUAL ARRAY NOT YET OPEN An attempt was made to use a virtual array before opening the corresponding disk file.	45
?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.	46
?LINE TOO LONG Attempt to input a line longer than 255 characters (which includes any line terminator). Buffer overflows.	47

Message Printed	ERR Value
%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)	48
%ARGUMENT TOO LARGE IN EXP Acceptable arguments are within the approximate range -89<arg<+88. The value returned is zero. (C)	49
%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)	50
%INTEGER ERROR An attempt to use a computed integer outside the range -32768<n<32767 can cause this error. 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)	51
?ILLEGAL NUMBER Integer overflow or underflow or floating point overflow can cause this error. 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.)	52
%ILLEGAL ARGUMENT IN LOG A negative or zero argument to LOG function can cause this error. Value returned is the argument as passed to the function. (C)	53
%IMAGINARY SQUARE ROOTS An attempt to take square root of a number less than zero can cause this error. The value returned is the square root of the absolute value of the argument. (C)	54
?SUBSCRIPT OUT OF RANGE An attempt is made to reference an array element beyond the number of elements created for the array when it was dimensioned.	55

Message Printed	ERR Value
?CAN'T INVERT MATRIX An attempt is made to invert a singular or nearly singular matrix.	56
?OUT OF DATA The DATA list was exhausted and a READ requested additional data.	57
?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.	58
?NOT ENOUGH DATA IN RECORD An INPUT statement did not find enough data in one line to satisfy all the specified variables.	59
?INTEGER OVERFLOW, FOR LOOP The integer index in a FOR loop attempted to go beyond 32766 or below -32767.	60
%DIVISION BY 0 The user program attempted to divide some quantity by zero. If no transfer is made to an error handler routine, a 0 is returned as the result. (C)	61
?NO RUN-TIME SYSTEM The referenced run-time system has not been added to the system list of run-time systems.	62
?FIELD OVERFLOWS BUFFER An attempt is made to use FIELD to allocate more space than exists in the specified buffer.	63
?NOT A RANDOM ACCESS DEVICE An attempt is made to perform random access I/O to a non-random access device.	64
?ILLEGAL MAGTAPE() USAGE The MAGTAPE function is used improperly.	65
?MISSING SPECIAL FEATURE User program employs a BASIC-PLUS feature not present on the given installation.	66
?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.	67

C.2 Non-Recoverable

Message Printed

?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 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 causes this error.

?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.

?ERROR TEXT LOOKUP FAILURE
An I/O error occurred while the system was attempting to retrieve an error message. Possible causes could be the device containing the system error file (ERR.SYS) is off-line, or the system error file contains a bad block.

?EXECUTE ONLY FILE
An attempt was made to add, delete or list a statement in a compiled 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 specified name 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.

Message Printed

?FNEND WITHOUT DEF
: An FNEND statement was encountered in the user program without a previous function call having been executed.

?FNEND WITHOUT FUNCTION CALL
: A FNEND statement was encountered in the user program without a previous DEF statement being seen.

?FOR WITHOUT NEXT
: A FOR statement was encountered in the user program without a corresponding NEXT statement to terminate the loop.

?ILLEGAL CONDITIONAL CLAUSE
: An incorrectly formatted conditional expression causes this error.

?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 a 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 specified FIELD variable is unacceptable.

?ILLEGAL FN REDEFINITION
: An attempt was made to redefine a user function.

?ILLEGAL FUNCTION NAME
: An attempt was made to define a function with a function name not subscribing to the established format.

?ILLEGAL IF STATEMENT
: An incorrectly formatted IF statement causes this error.

?ILLEGAL IN IMMEDIATE MODE
: The user issued a statement for execution in immediate mode which can only be performed as part of a program.

?ILLEGAL LINE NUMBER(S)
: A line number reference outside the range 1<n<32767 causes this error.

?ILLEGAL MODE MIXING
: String and numeric operations cannot be mixed.

Message Printed

?ILLEGAL STATEMENT
: An 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 causes this error.

?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.

?LITERAL STRING NEEDED
: A variable name was used where a numeric or character string was necessary.

?MATRIX DIMENSION ERROR
: An 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
An attempt is made to use one of the statement modifiers (FOR, WHILE, UNTIL, IF, or UNLESS) incorrectly. An OPEN statement modifier, such as a RECORDSIZE, CLUSTERSIZE, FILESIZE, or MODE option, is not in the correct order.

?NEXT WITHOUT FOR
A NEXT statement was encountered in the user program without a previous FOR statement having been seen.

?NO LOGINS
Message printed if the system is full and cannot accept additional users or if further logins are disabled by the system manager.

?NOT ENOUGH AVAILABLE MEMORY
An attempt is made to load a non-privileged compiled program which is too large to run, given the job's private memory size maximum. The program must be made privileged to allow it to expand above a private memory size maximum; or the system manager must increase the job's private memory size maximum to accommodate the program.

?NUMBER IS NEEDED
A character string or variable name was used where a number was necessary.

?1 OR 2 DIMENSIONS ONLY
An 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. The user is not logged into the system and 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.

Message Printed

?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.*

?REDIMENSIONED ARRAY
: Usage of an array or matrix within the user program has caused BASIC–PLUS to redimension the array implicitly.

?RESUME AND NO ERROR
: A RESUME statement was encountered where no error had occurred to cause a transfer into an error handling routine via the ON ERROR GOTO statement.

?RETURN WITHOUT GOSUB
: RETURN statement encountered in the user program without a previous GOSUB statement having been executed.

%SCALE FACTOR INTERLOCK
: An attempt was made to execute a program or source statement 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.

Message Printed

?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. An illegal verb or improper format error is 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.

C.3 Software Performance Report Guidelines

The Software Performance Report (SPR) forms enable you to report problems with DIGITAL software. Prior to submitting an SPR, ensure that the problem has not been corrected in the Release Notes or the Software Dispatch. To speed response and prevent processing delays with an SPR, you should describe the problem as completely as possible. The following list contains the minimal information that you should include on the SPR.

- Complete hardware configuration; including CPU type, system disk, amount and type of memory, hardware options (such as floating-point processor), and system peripherals.
- Monitor options present on the system, including math package and BASIC-PLUS options.
- Program name, version number, and edit level (generally found on line 1010 of the program listing), and any optional patches included in the program. Also include the account(s) under which the program failed and whether it is privileged or non-privileged.
- The PRIORITY and SWAP MAX under which the program was running.
- A terminal printout of any relevant command strings and input data.
- A list of any modifications made to the program.
- A listing of any applicable log files.
- If a crash dump is submitted, it must include system load map listing, "CORE DUMP OF MONITOR" output of the crash dump analysis program. Machine-readable submissions must include the CRASH.SYS file and installed monitor .SIL file. For problems related to run-time systems, include the .RTS file.

Appendix D
Radix-50 Character Set

Many items in RSTS/E, such as filenames 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 filename string scan SYS calls convert 3-character strings to Radix-50 format.

The following 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, 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.

Table D-1: Radix-50 Character/Position Table

Single Char. or First Char.		Second Character		Third Character	
A	003100	A	000050	A	000001
B	006200	B	000120	B	000002
C	011300	C	000170	C	000003
D	014400	D	000240	D	000004
E	017500	E	000310	E	000005
F	022600	F	000360	F	000006
G	025700	G	000430	G	000007
H	031000	H	000500	H	000010
I	034100	I	000550	I	000011
J	037200	J	000620	J	000012
K	042300	K	000670	K	000013
L	045400	L	000740	L	000014
M	050500	M	001010	M	000015
N	053600	N	001060	N	000016
O	056700	O	001130	O	000017
P	062000	P	001200	P	000020
Q	065100	Q	001250	Q	000021
R	070200	R	001320	R	000022
S	073300	S	001370	S	000023
T	076400	T	001440	T	000024
U	101500	U	001510	U	000025
V	104600	V	001560	V	000026
W	107700	W	001630	W	000027
X	113000	X	001700	X	000030
Y	116100	Y	001750	Y	000031
Z	121200	Z	002020	Z	000032
$	124300	$	002070	$	000033
.	127400	.	002140	.	000034
unused	132500	unused	002210	unused	000035
0	135600	0	002260	0	000036
1	140700	1	002330	1	000037
2	144000	2	002400	2	000040
3	147100	3	002450	3	000041
4	152200	4	002520	4	000042
5	155300	5	002570	5	000043
6	160400	6	002640	6	000044
7	163500	7	002710	7	000045
8	166600	8	002760	8	000046
9	171700	9	003030	9	000047

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.

Appendix E Device Handler Index

Table E-l contains a list of the handler indices for each device type used on the RSTS/E system. The handler index is an internal index into system device tables and is used by the system to identify device families. For example, the handler index is used in SPEC% functions to ensure that the correct device is operated upon.

Table E–1: Handler Index

Index	Device
0	All disks.
2	Terminals.
4	Dectape.
6	Line Printers.
8	Paper Tape Readers.
10	Paper Tape Punches.
12	Card Readers.
14	Magnetic Tapes.
16	Psuedo Keyboards.
18	Floppy Disks.
20	RJ2780.
22	Null Device.
24	DMC11/DDCMP Interface.
26	Auto-Dialers.
28	X-Y Plotters.
30	TU58 DECtape II.
32	KMC11.
34	IBM Interface.

Appendix F
Monitor Directives

The *RSTS/E System Directives Manual* contains information on Monitor directives for MACRO programmers. Many of these directives correspond to RSTS/E SYS calls described in Chapter 7 of this manual. Table F–1 contains the SYS call to FIP codes and the corresponding Monitor directives. For information on the use of these directives, refer to the *RSTS/E System Directives Manual.*

Table F–1: Monitor Directives

MACRO Mnemonic	Sys Call Code	Privileged Status	Function
UU.SPL	-28	no	Spooling.
UU.DMP	-27	yes	Snap shot dump.
UU.FIL	-26	yes	File utility functions.
UU.ATR	-25	no	Read or write attributes.
UU.CCL	-24	yes	Add/Delete CCL command.
(.FSS)	-23	no	Terminating filename string scan.
(.SET)	-22	yes	Set special run priority.
(.SET/.CLEAR)	-21	yes	Drop/Regain temporary privilege.
(.SET/.CLEAR)	-20	yes	Lock/Unlock job in memory.
UU.LOG	-19	yes	Set number of logins.
UU.RTS	-18	yes	Add run-time system.
		yes	Remove run-time system.
		yes	Load run-time system.
		yes	Unload run-time system.
		yes	Add resident library.
		yes	Remove resident library.
		yes	Load resident library.
		yes	Unload resident library.

(continued on next page)

Table F-1: Monitor Directives (Cont.)

MACRO Mnemonic	Sys Call Code	Privileged Status	Function
UU.NAM	-17	no	Name run-time system.
UU.DIE	-16	yes	System shutdown.
UU.ACT	-15	yes	Accounting dump.
UU.DAT	-14	yes	Change system date/time.
UU.PRI	-13	yes	Change priority/run burst/job size.
UU.TB2	-12	no	Get Monitor tables - part II.
UU.BCK	-11	yes	Change file backup statistics.
(.FSS)	-10	no	Filename string scan.
UU.HNG	-9	yes	Hangup a dataset.
UU.FCB	-8	no	FCB/DDB information.
–	-7	no	CTRL/C trap enable.
UU.POK	-6	yes	Poke memory (1,1) only.
(.SPEC)	-5	yes	Broadcast to a terminal.
(.SPEC)	-4	yes	Force input to a terminal.
UU.TB1	-3	no	Get Monitor tables - part I.
UU.NLG	-2	yes	Disable logins.
UU.YLG	-1	yes	Enable logins.
UU.PAS	0	yes	Create user account.
UU.DLU	1	yes	Delete user account.
UU.CLN	2	yes	Clean up a disk pack.
UU.MNT	3	yes	Disk pack and terminal status.
UU.LIN	4	yes	Login.
UU.BYE	5	yes	Logout.
UU.ATT	6	yes yes	Attach. Reattach.
UU.DET	7	yes	Detach.
UU.CHU	8	yes yes yes	Change password/quota. Kill job. Disable terminal.
UU.ERR	9	no	Return error message.
UU.ASS	10	no	Assign/Reassign device.
.ULOG		both	Enter user logical.

(continued on next page)

Table F-1: Monitor Directives (Cont.)

MACRO Mnemonic	Sys Call Code	Privileged Status	Function
UU.DEA	11	no	Deassign device or remove user logical.
UU.DAL	12	no	Deassign all devices.
UU.ZER	13	both	Zero a device.
UU.RAD	14	both	Read or read and reset accounting data.
UU.DIR	15	no no	Directory look up on index. Special magnetic tape directory look up.
UU.TRM	16	both	Set terminal characteristics.
UU.LOK	17	no no	Disk directory look up on filename. Disk wildcard directory look up.
(.MESAG)	18	both yes yes	Send a message. Declaring a receiver/receiving a message. Remove from receiver table.
UU.CHE	19	yes	Enable/Disable disk caching.
UU.CNV	20	no	Date and Time conversion.
UU.SLN	21	yes	System logical names.
(.MESAG)	22	both	Message send/receive.
UU.SWP	23	yes	Add/Remove system files.
UU.JOB	24	yes	Job creation.
UU.PPN	25	no	Wildcard PPN look up.
UU.SYS	26	no	Return system status information.

Index

Where more than one page number appears for an entry, the defining entry is in bold type.

RSTS/E
Programming Manual
AD-2726C-T1

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code or Country ______

— — — — —Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Commercial Engineering Publications MK1-2/ H3
DIGITAL EQUIPMENT CORPORATION
CONTINENTAL BOULEVARD
MERRIMACK N.H. 03054

— — — — Do Not Tear - Fold Here and Tape — — — — — — — — — — — — — — — — — —

Along Dotted Line