RSTS/E
Programming Manual

Order No. AA-EZ09A-TC
Including AD-EZ09A-T1,T2

June 1987

This manual describes RSTS/E special programming techniques. It contains information on device-dependent features and the use of system function calls.

OPERATING SYSTEM AND VERSION:	RSTS/E	V9.4
SOFTWARE VERSION:	RSTS/E	V9.4

digital equipment corporation, maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

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

The following are trademarks of Digital Equipment Corporation:

digital™	DECUS	Rainbow
	DECwriter	ReGIS
DATATRIEVE	DIBOL	RSTS
DEC	FMS-11	RSX
DECmail	LA	RT
DECmate	MASSBUS	UNIBUS
DECnet	PDP	VAX
DECsystem-10	P/OS	VMS
DECSYSTEM-20	Professional	VT
DECtape	Q-bus	Work Processor
		WPS-PLUS

CONTENTS

CHAPTER 3 Line Printer

CHAPTER 4 Terminals

PART II System Function Calls and Programming Hints

CHAPTER 7 SYS System Function Calls

FIGURES

TABLES

Preface

Objectives

This manual describes RSTS/E programming techniques. It:

- Explains how to optimize the use of devices on RSTS/E
- Describes system function calls to the RSTS/E monitor
- Provides general information and programming hints for the system programmer

Audience

This manual is for BASIC-PLUS, BASIC-PLUS-2, and MACRO programmers. It assumes that you know how to program in one of these languages and are familiar with RSTS/E system concepts and features.

If you program in BASIC-PLUS or BASIC-PLUS-2, this manual contains all the information you need to use device-dependent features and system function calls. If you program in MACRO, however, you will need to use this manual as a companion to the *RSTS/E System Directives Manual*.

Document Structure

Part I, Devices, contains six chapters. Each chapter describes programming techniques for a different type of device:

Chapter 1	Describes file-structured and non-file-structured disk and flexible diskette operations. It also describes RSTS/E system files and privileges.
Chapter 2	Describes file-structured and non-file-structured magnetic tape operations and explains how to process DOS- and ANSI-labeled tapes.
Chapter 3	Describes system features for controlling line printers.
Chapter 4	Describes system features for controlling terminals, such as echo control and multiterminal service. It also describes pseudo keyboards.
Chapter 5	Describes DECtape, paper tape, and card readers.
Chapter 6	Describes the DMC11/DMR11 interprocessor link.

Part II, System Function Calls and Programming Hints, contains four chapters:

Chapter 7 Describes system function calls available to BASIC-PLUS and BASIC-PLUS-2 programmers. These calls let you communicate with the RSTS/E monitor, perform special I/O functions, and set terminal and job characteristics. Although the call descriptions are tailored for BASIC programmers, MACRO programmers can consult this chapter for a detailed description of the corresponding monitor directives.

Chapter 8 Describes system function calls for local message send/receive operations. As in Chapter 7, the call descriptions are tailored for BASIC programmers but are intended for use by MACRO programmers as well.

Chapter 9 Describes the system function call for a Print/Batch Services (PBS) request.

Chapter 10 Contains system programming hints. It describes the CCL facility and explains how the monitor handles the SLEEP and conditional SLEEP statements.

Chapter 11 Describes Ethernet and the commands for using it.

This manual also has seven appendixes:

Appendix A Describes magnetic tape label formats for DOS and ANSI tapes and explains how RSTS/E initializes the two types of tapes.

Appendix B Lists card codes.

Appendix C Lists RSTS/E and BASIC-PLUS error messages.

Appendix D Summarizes the Radix-50 and ASCII character sets.

Appendix E Lists device handler indexes.

Appendix F Lists the monitor directives that correspond to the BASIC-PLUS system function calls.

Appendix G Describes the use of parameters and other features of the send/receive calls that are specific to an EMT logging program.

Related Documents

The *RSTS/E System User's Guide* describes RSTS/E system concepts, and explains how to work with files and devices.

The *RSTS/E Utilities Reference Manual* describes the use of RSTS/E system programs.

The *BASIC-PLUS Language Manual* describes how to program in BASIC-PLUS.

The *RSTS/E System Directives Manual* describes monitor directives available to MACRO programmers.

See the *RSTS/E Documentation Directory* for more information on RSTS/E manuals.

Conventions

This manual uses the following conventions:

< > Angle brackets enclose essential elements of the item being described. For example, you must supply an expression in the statement:

```
SLEEP <expression>
```

[ ] Square brackets indicate an optional element or a choice of one element among two or more optional elements. For example, the CCL DETACH switch has the form:

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

The required part of the switch is /DET.

CTRL/x This symbol indicates a control key combination, such as CTRL/U or CTRL/O. To enter a control key combination, hold the CTRL key down while you press the indicated key.

All examples in this manual are written to execute in BASIC-PLUS EXTEND mode unless otherwise noted. If you enter them at your terminal, remember to press the RETURN or LINE FEED key after each command, statement, or program line.

Summary of Technical Changes for RSTS/E V9.0

Version 9.0 is a major release of the RSTS/E operating system. This manual contains support for the following new features:

- Multiple privileges: A set of privileges now control access to system functions.
- Print/Batch Services (PBS): A new spooling package provides improved print and batch capability. V9.0 still supports the OPSER-based spooling package.
- Quota specification: You can specify logged-in, logged-out, and detached account quotas. In addition, the system now performs logged-in quota checking. V9.0 still supports the old quota format.
- Long passwords: You can specify an account password as 14 ASCII characters. V9.0 still supports the old format password.
- Expiration date: You can specify an account expiration date
- 8-bit character support: The system provides transparent handling of 8-bit characters.
- Virtual disk: A logical device that supports the storage of temporary data within system memory.
- Nine character logical names.
- New terminal features.

RSTS/E V9.0 supports the following new hardware: PDP-11/73 and PDP-11/84 systems, DHU11/DHV11 multiplexer, VT220/240 and LA200 terminals, RD52 disk, RC25 disk, TK25 tape, LN03 laser printer, and LQP03 printer. See the *RSTS/E V9.0 Release Notes* for a complete list of supported hardware.

RSTS/E V9.0 no longer supports the following hardware: RF11 disks, RS03/RS04 disks, RK05 as a system disk, VT05 and VT50 terminals, TU58 DECtape-II. See the *RSTS/E V9.0 Release Notes* for a complete list of unsupported hardware.

System Function Call Changes

V9.0 has five new SYS system function calls. Many existing calls have expanded functions. The following is a summary:

New System Function Calls

Set/Clear/Read Current Privileges (SYS 28)
: This call reads the current privilege mask and selectively sets and/or clears bits in it.

Third Party Privilege Check (SYS 31)
: Server programs such as spoolers use this call to perform privilege checks on users who request the service.

Check Access Function (SYS 32)
: This call performs a variety of privilege checking functions. It checks file access rights, converts a privilege mask to names, and converts privilege names to mask.

Open Next Disk File (SYS 33)
: This call opens a disk file or a series of disk files matching a wildcard specification.

Set Device Characteristics/Set System Defaults (SYS 34)
: This call sets certain device characteristics, line printer characteristics, and the following system defaults: date, time, magnetic tape labeling, and magnetic tape density. The call also loads and removes certain monitor overlay codes.

New Subfunctions of Existing System Function Calls

Send Print/Batch Services Request (Message Send/Receive, SYS 22)
: This call allows an application program to issue a request for print/batch services. It has the same range of capabilities as the DCL PRINT or SUBMIT commands.

Send Local Data Messages With Privileges (Message Send/Receive, SYS 22)
: This subfunction provides a method for a program to tell another program about a job's current privileges.

Change Quota/Expiration Date/Password (SYS 8)
: This call is restructured to accomodate quota checking and password changing in both the new and old format. It also lets you specify an account expiration date.

Read Pack Attributes, Read/Write/Delete Account Attributes (SYS -25)
: These subfunctions expand your ability to manipulate accounting information.

List Logical Names (SYS 21)
In addition to the new List subfunction, this call now supports nine-character logical names.

Set Terminal Characteristics - Part II (SYS 16)
This subfunction expands the range of settable terminal characteristics. The call now supports terminal settings for 8-bit characters, auto-baud, CTRL/C and CTRL/X, terminal capability flags, separate control characters, and terminal type.

Create Dynamic Region (SYS -18)
This subfunction creates both named and unnamed dynamic regions of memory.

Verify Password (SYS 4)
This addition to the Login call allows an application to verify a password without having to first logout.

Return Job Status (SYS 26)
A new subfunction returns information on I&D space as well as the current privilege mask.

Deleted System Function Calls

Clean a Disk (SYS 2)
The DCL MOUNT command now performs this function.

Declare Default Keyboard Monitor (subfunction of SYS -18)
In V9.0, DCL is always the system default keyboard monitor.

Other System Function Call Changes

All SYS calls that handle quota, password, or privilege checking now support the V9.0 format. Among the calls affected are SYS calls 0, 5, 6, 14, and 24.

There are also minor changes to SYS calls -29, -28, -26, 10, 15, and 17.

Other Technical Changes

For line printers, new MODE values let you specify a form length up to 255 lines, disable escape sequences, and set NOWRAP. For individual output steps, new RECORD modifiers let you truncate long lines and enable binary output. Two new SPEC% functions return page counter and line position information.

For magnetic tape, a new MAGTAPE function lets you write an end-of-volume (EOV) tape mark, providing support for multivolume ANSI tape processing. Another new MAGTAPE function provides support for error handling in asynchronous I/O requests.

For terminals, you can now use a new MODE value and RECORD modifier to correctly handle a 256-character set. In addition, a new RECORD modifier lets you force program input from a terminal even when the program is running under the control of a DCL command file.

For pseudo keyboards, a new SPEC% function returns the exit status of a controlled job.

Summary of Technical Changes for RSTS/E V9.1

Version 9.1 is a mini-release of the RSTS/E V9.0 operating system. This manual contains support for the following new features:

All references to the primary run-time system have been changed to the default keyboard monitor. Specifically, pages 1-3, and 1-8.

TMPPRV privilege description has been added to Table 1-4.

Default Characteristics for the RD53 disk have been added to Table 1-8.

System Density Values for the TK50 magnetic tape drive have been added to Table 2-2.

A new subfunction code has been added to the MAGTAPE and SPEC% calls. This function can be used to set the density of a tape drive, or to return density information about a drive. See pages 2-24, 2-25, and 2-36.

Two new subfunction codes have been added to the Disk Pack Status SYS call (F0=3). These functions can be used to load the Storage Allocation Table (SAT) of a disk into memory or unload a SAT from memory. See pages 7-160 to 7-164.

The data passed and returned for the Set System Defaults Function of the Set Device Characteristics and System Defaults SYS call (F0=34) have been changed. See page 7-311.

Summary of Technical Changes for V9.4

Significant changes to the *RSTS/E Programming Manual* for V9.4 are:

Ethernet has been added. See Chapter 11 for an overview of Ethernet and a description of how to use its local area networking features on RSTS/E.

PART I
Devices

Chapter 1

System Structure and Disk Operations

Disks are file-structured, random access devices. They are the fastest, most reliable, and most durable type of peripheral device.

RSTS/E is a disk-based system. During timesharing, some parts of the monitor and run-time system code are always in memory; other parts are on the system disk and are loaded into memory only when needed. The system disk also stores system programs and user files.

Because the RSTS/E system is built around disks and their characteristics, this chapter differs from other chapters on peripheral devices in this manual. Besides describing both file-structured and non-file-structured disk operations, it also describes how RSTS/E system accounts are set up and how RSTS/E handles privileges. This chapter also describes flexible diskettes and the "null device," a software structure available on all RSTS/E systems for debugging and for creating a buffer without tying up a physical device.

System Accounts

RSTS/E systems have two accounts that are essential to system operation: the system library account and the system account. The system library account, [1,2], stores a library of system programs and message and control files. This account must be present on the system disk. The system account, [0,1], contains RSTS/E monitor files and routines that are critical to system operation.

The following sections explain these two accounts in detail.

System Library Account [1,2]

During system installation, the initialization procedure creates the system library account [1,2] on the system disk. The system program installation procedure populates the account with system programs. This section briefly describes the contents of account [1,2]. See the *RSTS/E System Installation and Update Guide* for a directory listing of the account.

The system library stores many of the system programs that are available to general and privileged users. It also contains text files used by system programs.

During normal system start-up or automatic crash recovery, the START option accesses the system library automatically. The console keyboard is logged in automatically under account [1,2]. Then the system invokes the START.COM file in account [0,1] as a DCL command file. One of the steps in the system start-up procedure runs the ERRCPY program. Depending on the contents of the start-up command file, other programs may be started up in account [1,2] as well, or the system manager may elect to run any of these programs in some other account.

System Account [0,1]

During system installation, the initialization procedure creates the system account [0,1] on the system disk. The procedure creates two files required for all RSTS/E disks and stores them in [0,1]: the storage allocation file SATT.SYS and the bad block file BADB.SYS.

Account [0,1] on the system disk also contains files used for system operation. During system installation, the system copies the necessary files into [0,1]. See the *RSTS/E System Installation and Update Guide* for a directory listing of the account. Some of the most important files are:

- INIT.SYS, the system initialization code
- SWAP.SYS, the primary swapping file
- CRASH.SYS, the crash dump data file
- A file with the file type .SIL, the monitor code
- DCL.RTS, the system default keyboard monitor
- RT11.RTS, a required auxiliary run-time system
- ERR.ERR, the error message text
- CSPLIB.LIB, the system program resident library

The following sections describe the RSTS/E system files.

Allocating Disk Storage Space

RSTS/E uses the SATT.SYS file to control 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 Storage Allocation Table (SAT). 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 512-byte blocks of storage on the disk. The cluster size defines how many contiguous 512-byte blocks are contained in the cluster. RSTS/E defines cluster sizes for disks, directories, and files.

Table 1-1 presents the types of clusters and related information.

Table 1-1: Valid Cluster Size Ranges

Cluster Size	Minimum Size	Maximum Size	When Defined
Pack (for any disk)	Device Cluster Size (see Table 1-8)	16	At initialization time with DSKINT option, or on line with the DCL INITIALIZE command.
Directory	Pack Cluster Size	16	At creation of the directory with either the DSKINT initialization option, CREATE/ACCOUNT command, or SYS system function.
File	Pack Cluster Size	256	At creation of the file with either an OPEN or OPEN FOR OUTPUT statement, or the DCL CREATE or COPY command. Specify cluster size with the CLUSTERSIZE option. Note that when you specify a negative cluster size, the system uses either the absolute value of the argument specified or the pack cluster size, whichever is greater.

The system manager specifies the disk cluster size either during disk initialization (DSKINT) or on line with a qualifier to the DCL INITIALIZE command . The pack cluster size defines the minimum number of contiguous 512-byte blocks that a cluster comprises on a specific disk; thus, the extent of contiguous space each bit represents in the SAT. A pack cluster size of 1 means that one 512-byte block of storage is allocated for each bit set to 1. A pack cluster size of 2 means that two contiguous 512-byte 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. See table 1-8 for a list of disk device cluster sizes.

The pack cluster size affects the efficiency of storage space allocation. A large size improves access time to programs and files

but may waste disk space. For example, if the pack cluster size is 16, the system allocates one cluster of 16 contiguous blocks to a one-block file: 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 that best fits the type of processing and the access requirements of the local installation.

One processing consideration is the use of data caching on the system (see the section "Caching Control"). While the pack cluster size is set during disk initialization and the cache cluster size can be set and changed during timesharing, the relationship between the two affects the optimal use of the cache. For example, if the pack cluster size and file cluster size are both 4 and you specify a cache cluster size of 8 (see the SET CACHE command in the *RSTS/E System Manager's Guide*, or SYS Call 19, Enabling and Disabling Disk Caching), 4 blocks in the cache contain your file's data and 4 may contain unrelated data. Therefore, if you plan to use data caching on your system, the pack cluster size that the system manager specifies during disk initialization should be equal to or greater than any cache cluster size you specify during timesharing.

The User File Directory (UFD) has a defined directory cluster size. Its minimum value is the pack cluster size. The system manager specifies the cluster size during account creation. A directory cluster size must be a power of 2 up to a maximum of 16 and must be greater than or equal to the pack cluster size. 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 expands to catalog files and can occupy a maximum of seven clusters.

The directory 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 directory 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 is 72 for a UFD cluster size of 1 and the maximum UF is 1157 for a UFD cluster size of 16. Note that system performance is maximized when the UFD contains fewer files.

Bad Block File

The bad block file BADB.SYS is the mechanism which the system manager uses to remove unreliable storage blocks on system and nonsystem disks from use. The DSKINT option or the DCL INITIALIZE command creates BADB.SYS in account [0,1]. 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, more 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 (see the *RSTS/E Installation and Update Guide*), the manager can add newly discovered bad blocks to BADB.SYS. Once the system allocates a bad block to BADB.SYS, it cannot be deallocated.

Note that MSCP disk controllers provide their own built-in handling of bad disk blocks. This is transparent to the system; the disk always appears to have the full number of good data blocks. This applies to RA80, RA81, RA60, RC25, RD51, and RD52 disks.

System Overlay File

The OVR.SYS file contains certain monitor code that resides on disk, not in memory. The system loads this code into memory on demand and overlays a certain part of the monitor. The monitor Save Image Library (SIL) normally contains the overlay code. The system achieves optimum efficiency when this code resides on the center of a fast-access disk.

If the system disk is not a fast-access disk, the system manager can use the DCL INSTALL/OVERLAY_FILE command to create a separate, contiguous file that contains the overlay code. The manager can optimally position this file on a fast disk. At the start of time-sharing operations, the system manager can add the overlay file 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 Directory File

The BUFF.SYS file is used in DECtape processing. When you open a file on DECtape, the system writes the directory of the DECtape to the file BUFF.SYS. The BUFF.SYS file requires three 512-byte blocks for each

DECtape drive on the system. Any updates to the DECtape directory 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 written back to the DECtape when the last file open on the DECtape unit is closed or any output file is closed.

During system installation, the system automatically creates the BUFF.SYS file if it is needed.

Monitor Save Image Library File

All 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 a file type of .SIL. Multiple monitor files can reside on the system disk but the system only installs one such file at a time. The system marks the installed monitor file as nondeletable and loads the file from disk to memory when time-sharing operations begin.

Error Messages File

The ERR.ERR file contains the system error messages. DIGITAL distributes ERR.ERR with each RSTS/E system. ERR.ERR must exist in account [0,1] on the system disk.

The DCL INSTALL/ERROR_FILE command allows the system manager to create a separate contiguous file and position it on any disk. The standard name for this file is ERR.SYS. The system achieves optimum efficiency when this code resides on a fast-access disk. At the start of time-sharing operations, the system manager can add this separate file to 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.

Saving Information After a Crash

The system uses the file CRASH.SYS to save a dump of the read/write area of the monitor and the extended buffer pool (XBUF) at the time of a system crash.

INIT.SYS automatically creates the CRASH.SYS file on the system disk during system start-up. If INIT.SYS cannot find sufficient contiguous

disk space to create CRASH.SYS, it prints a warning message before starting the system.

The size of CRASH.SYS depends on the size of the monitor read/write area and XBUF. The monitor read/write area size varies according to the hardware and software configuration but is between 48 and 96 blocks. To estimate the number of blocks needed for XBUF, use the formula:

```
Size of XBUF in K words * 4
```

Run-Time System Files

The account [0,1] on the system disk must contain at least one file with a file type of .RTS. This file is the default keyboard monitor and is automatically loaded into memory by the monitor at the start of timesharing. The default keyboard monitor must reside on the system disk because that disk is the only one logically mounted at system start-up time.

DCL.RTS is the system default keyboard monitor. In addition, RT11.RTS is also required on the system. The system manager can add auxiliary run-time systems (other files with .RTS file types in account [0,1]).

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

System Program Resident Library

Account [0,1] on the system disk contains the resident library CSPLIB.LIB. Because nearly all system programs use CSPLIB.LIB, this resident library is required on the system. CSPLIB.LIB is automatically installed during system start up.

CSPLIB.LIB is a floating resident library. See the system function call, Manipulate Run-Time System, Resident Library, Dynamic Region (SYS -18), for more information about floating libraries.

Initialization Code

The INIT.SYS file contains the system initialization code. INIT.SYS resides in 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 that performs consistency checks on system software and hardware. It allows the system manager to:

- Initialize and format disks
- Install patches
- Enable and disable device controllers and units
- Manipulate files in account [0,1] on both system and non-system disks
- Change some default timesharing characteristics

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 timesharing, the RSTS/E monitor code replaces the initialization code in memory.

Swapping Storage

Nonresident jobs on RSTS/E are kept in predefined areas on disk called swap files. RSTS/E provides four distinct swap files: SWAP.SYS, SWAP0.SYS, SWAP1.SYS, SWAP3.SYS. Swap file number 2, named SWAP.SYS, is required on all systems; the other files are optional. SWAP.SYS must reside on the system disk.

During system installation, INIT.SYS automatically creates the SWAP.SYS file in account [0,1] at a size large enough for 1 job. Later on in the system installation, the system manager can create a SWAP1.SYS file at a size large enough to hold the rest of the jobs on the system. Or, the system manager can later create multiple swap files (up to a total of 4) to provide swap space for all jobs. The system manager can locate some or all of these files on disks other than the system disk, preferably on high speed disks that do not contain frequently accessed files. See the *RSTS/E System Installation and Update Guide* for details.

RSTS/E uses swap files in a predefined way. For example, the system stores a highly interactive job that must be removed from memory in the lowest numbered file available. The system searches for an empty space starting at the lowest numbered active file. On the other hand, a job with infrequent activity is stored in the highest numbered file available. Such relatively inactive jobs are those that sleep until an event occurs. The system error logging program ERRCPY is an example of a relatively inactive job.

A swap file can be either a file or an entire device, for example, a high speed disk. The best device to use for swap files on a system depends on the types of devices available and the amount of data swapped.

Table 1-2 shows the approximate amount of time (in seconds) needed to transfer different size job images for various types of disks. Actual times will be longer if the disk is accessed in other ways, for example to read user file data.

Table 1-2: Swap Times

Disk	Job Size (in words)				
	8K	16K	28K	32K	64K
RL01/02	.10	.13	.18	.19	.32
RK05	.16	.25	.38	.43	.78
RK06/07	.08	.11	.15	.17	.29
RP02/03	.10	.16	.24	.27	.50
RP04	.06	.08	.11	.12	.20
RP05/06	.06	.08	.11	.12	.20
RM02	.06	.08	.11	.12	.20
RM03/05	.05	.06	.08	.09	.14
RM80	.05	.06	.08	.09	.14
RA60	.06	.07	.08	.08	.11
RA80	.05	.06	.08	.09	.14
RA81	.04	.05	.06	.07	.10
RC25	.04	.06	.07	.08	.13
RD51	.11	.14	.17	.19	.29
RD52	.08	.10	.14	.15	.26

Calculate the swap times for each disk by using the formula:

$$\text{Swap time} = \text{Avg access time} + \frac{\text{Job size} * 2}{\text{Transfer speed}}$$

where:

Average access time	measured in seconds, is defined as the sum of the average seek time and the average latency time.
Transfer speed	is measured in kilo-bytes per second (KB/S).
Job size	is measured in kilo-words (KW).

See the appropriate *Peripherals Handbook* for the average access time and transfer speed of each disk.

When a file is used as a swap file, the system manager can further reduce the swap time by using the /POSITION switch on the file specification to position the file in the middle of the disk. This minimizes the time required for positioning the read/write heads. On systems with multiple disks, the system manager can position two files on separate drives to take advantage of overlapped seeks.

A swap file other than file 2 (SWAP.SYS) is dynamic. The system manager adds files at the start of timesharing to allow the maximum number of jobs to run. During timesharing, a swap file can be removed and added again as another device or file. Dynamic addition and removal of swap files allows timesharing to continue when hardware problems on a device being used for swapping would normally require discontinuing system operation.

System Account [0,1] on Nonsystem Disks

The system account [0,1] on a nonsystem disk initially contains two required files: SATT.SYS and BADB.SYS. The DSKINT initialization option or the DCL INSTALL FILE command similarly creates these files for nonsystem disks as for the system disk. Account [0,1] on a nonsystem disk, either public or private, can contain other optional system files.

The REFRESH initialization option manipulates system files in account [0,1] on a nonsystem disk as well as on the system disk. The following DCL commands perform the same operations: the /ERROR_FILE, /SWAP_FILE, and /OVERLAY_FILE qualifiers of the INSTALL and REMOVE commands; the SET FILE/[NO]DELETABLE command. See the *RSTS/E System Managers Guide* for more information about these commands.

Both the REFRESH option and DCL commands can create and position contiguous files (such as a swap file or the overlay file) on a nonsystem disk. They can also mark files in account [0,1] as nondeletable. Note that only REFRESH can add blocks to the BADB.SYS file. Nonsystem disks can also contain auxiliary run-time system files.

Storage of Accounting Data

This section describes how accounting data is stored on system and nonsystem disks. It describes:

- Accounting data on the system device.
- Accounting data on nonsystem disks.

Accounting Data on the System Device

Project-programmer numbers (PPN) and passwords control access to the RSTS/E system. The system manager, or anyone who has sufficient privilege (GACNT for group, WACNT for all), creates a new account by using the CREATE/ACCOUNT command (see the *RSTS/E System Manager's Guide*). The manager enters the PPN and password for the new account, along with other information, to allow a user access to system facilities.

The new account information is stored on the system device. During account creation, the system manager has the option to preextend and position the UFD (see SYS Call 0, Create User Account). By default, the system preallocates one cluster for the UFD. The UFD is related directly to the user's account and contains information about the files created under that account number.

The system disk structure contains information about all UFDs (accounts) on the system. When a user tries to gain access to the RSTS/E system by giving an account and password, the system program LOGIN checks whether the PPN and password given match one stored on disk. If so, the system allows access.

Besides the LOGIN program, other system commands and programs also access the account information. For example, the SHOW ACCOUNT command references the accumulated system accounting information. The system manager uses the SET/ACCOUNT command to reset this accounting data or change certain parameters such as disk quota. The LOGOUT system program references the disk quota information.

Table 1-3 lists the account information that the system keeps for each account.

Table 1-3: Account Information Stored on the System Device

Type	Description	Explanation
Identification	Project-programmer number (account)	The PPN has the format [n,m] where n and m are decimal numbers that identify the user.
	Password	6 letters and/or digits (old format). 14 ASCII characters (new 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 (log-in time)	Number of minutes the user has been connected to the system through a terminal or remote line.
	Kilo-core-ticks (KCTs)	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 and System Resource Usage	Quota	Number of 512-byte blocks the user is allowed to retain. Types of quotas include logged-out, logged-in, job, detached-job, message, and RIB.

Using SYS system function calls, users who have GACNT or WACNT privilege can write programs that access the accounting information. See the description of the system function calls in Chapter 7.

Accounting Data on Nonsystem Disks

The system disk exists in what is called the public structure. The system manager can add additional disks to the public structure or add them as private disks. Disks other than the system disk are called nonsystem disks. Each disk added to the system also contains its own directory structure, which is created when the system manager initializes the disk. A nonsystem disk initially contains UFD information for account [0,1] as well as storage information.

Accounts on public disks are treated differently from accounts on private disks. RSTS/E allocates space for a user's file in the public structure on the disk that has the most free space. If the user's account does not yet exist on the disk with the most free space, the account number is added dynamically to that disk and a UFD is created for the user on that disk. A user cannot create a file on a private disk unless the account number already exists on that disk. The system manager or a sufficiently privileged user grants access to a private disk by entering the account information on the desired disk with the CREATE/ACCOUNT command.

Privileges

The system manager must have a way to prevent general access to activities that can damage the system. In previous releases, RSTS/E allowed the system manager to divide users into privileged and nonprivileged groups. Nonprivileged users were restricted to activities that could cause no system damage. Privileged users had access to all activities.

The V9.0 multiple privileges feature gives the system manager finer control over access to activities. Now the system manager can limit the user's access to just those activities suitable to the user's job. Multiple privileges gives the system manager a tool to enhance system performance, security, and more easily delegate certain operations.

Multiple Privileges

The multiple privilege feature groups similar system functions into sets and defines a privilege to control access to each set of functions. A group of 34 privileges govern the entire set of RSTS/E system functions. The privileges given to an account determine the range of functions available to the user. Some privileges apply to very specific functions; others control functions within broader classes of system use.

Table 1-4 summarizes the RSTS/E privileges.

Table 1-4: RSTS/E Privileges

Privilege	Description
DATES	Change system date/time and file dates.
DEVICE	Access restricted devices.
EXQTA	Exceed quotas or memory maximum. (Not usually given to users; used by privileged programs.)
GACNT	Perform accounting operations on accounts in the user's group.
GREAD	Read or execute any file in the user's group, regardless of protection code.
GWRITE	Write, delete, create, or rename any file in the user's group, regardless of protection code.
HWCFG	Set hardware configuration parameters; for example, set terminal characteristics.
HWCTL	Control devices; for example, disable a device or hang up a dial-up line.
INSTAL	Install run-time systems, swap files, and resident libraries.
JOBCTL	Manipulate other jobs; for example, detach or kill a job.
MOUNT	Mount or dismount disks other than NOSHARE.
PBSCTL	Control Print/Batch Services (PBS); for example, turn servers on or off, and change printer forms.
RDMEM	PEEK at memory. (Not usually given to users; used by privileged programs.)
RDNFS	Read a disk non-file-structured.
SEND	Broadcast to terminals and send messages to restricted receivers.
SETPAS	Change your own password.

Table 1-4: RSTS/E Privileges (Cont.)

Privilege	Description
SHUTUP	Shut down the system.
SWCFG	Set software configuration parameters; for example, installation name.
SWCTL	Control software components; for example, turn DECnet on and off.
SYSIO	Perform restricted I/O operations; for example, gain write access to files in account [0,*], or set the privilege bit on executable files.
SYSMOD	Perform functions that could easily modify the system; for example, poke memory.
TMPPRV	Set privilege bit (128) in the protection code of an executable program.
TUNE	Control system tuning parameters; for example, caching or job priority.
USER1-8	Available for customer applications. Not used by RSTS/E.
WACNT	Perform accounting operations on any account.
WREAD	Read or execute any file regardless of protection code.
WRTNFS	Read/write a disk non-file-structured.
WWRITE	Write, delete, create, or rename any file regardless of protection code. (For [0,*] accounts, SYSIO is required in addition to WWRITE.)

Classes of System Functions

Most system activities fall into two general classes:

- Account Management Activities
- File Access Activities

The next two sections describe these two classes of system activities and discuss the privileges that control them.

Account Management Activities

A user accesses a computer through an account. The individual account is a member of the "group," which contains all accounts with the same project number. The group, in turn, is a subset of the "world," which contains all accounts on the system. Account management activities include creating and deleting accounts, as well as changing passwords, disk quotas, and expiration dates.

The following privileges control account management:

GACNT — Group Account Management -- Grants account management privileges within the user's group.

WACNT — World Account Management -- Grants account management privileges for all accounts.

SETPAS — Set Password -- Allows changing one's own password.

Table 1-5 outlines the account management activities and the privileges required to perform them.

Table 1-5: Account Management Privileges

Activity	Self	Group	World
Create/delete account	GACNT or WACNT (for nonsystem disks *)	GACNT or WACNT	WACNT
Set account parameters	GACNT or WACNT	GACNT or WACNT	WACNT
Set password	SETPAS or GACNT or WACNT	GACNT or WACNT	WACNT
Read account data/parameters	Always allowed, except password	GACNT or WACNT	WACNT
Read/reset account data	GACNT or WACNT	GACNT or WACNT	WACNT

* Create does not apply to the system disk; you cannot delete your own account.

File Access Activities

Users routinely access files. The user creates some files, which reside in the individual's account. Other files reside in the accounts of other users or in system accounts. File access activities include: creating, deleting, renaming, reading, writing, and executing files.

Both the protection code of the file and the privileges granted to the user can affect whether the system grants or denies file access.

On a system with equal privileges granted to all users, protection codes control the operations that a user can perform on a file. The SET PROTECTION command (or the /PROTECTION switch in the RSTS/E file specification) passes a value to the system that sets bits in the protection code byte. When a bit is set, the the system prohibits activity named by that bit.

Table 1-6 shows the value and meaning of each protection code bit.

Table 1-6: RSTS/E File Protection Codes

If Executable Bit Not Set

128	64	32	16	8	4	2	1
Priv	Exe (0)	Write World	Read World	Write Group	Read Group	Write Owner	Read Owne

If Executable Bit Set

128	64	32	16	8	4	2	1
Priv	Exe (1)	Read, Write World	Exe World	Read, Write Group	Exe Group	Read, Write Owner	Exe Owne

Certain privileges also govern file access activities. Some privileges override protection codes completely. The following privileges grant a user the right to perform certain file access activities, regardless of protection codes:

GREAD — Group Read -- Read the data in any file within the group. Also, execute a program, if the executable bit is set.

WREAD — World Read -- Read the data in any file in on the system. Also, execute a program, if the executable bit is set.

GWRITE — Group Write -- Modify, extend, or delete the data in any file within the group.

WWRITE — World Write -- Modify, extend, or delete the data in any file on the system.

Table 1-7 summarizes the file access activities and the rules that govern file access.

Table 1-7: File Access Privileges

Function	Self	Group	World
Read	Yes, if protection code permits, or GREAD or WREAD	GREAD or WREAD or protection code permit	WREAD or protection code permit
Write/Delete	Yes, if protection code permits, or GWRITE or WWRITE	GWRITE or WWRITE or protection code permit	WWRITE or protection code permit (and SYSIO if account [0,*])
Execute	Yes, if protection code permits, or GREAD or WREAD	GREAD or WREAD or protection code permit	WREAD or protection code permit
Create/Rename /Zero	Yes	GWRITE or WWRITE	WWRITE (and SYSIO if account [0,*])

Multiple Privilege Masks

The system manager assigns a certain set of privileges to each account. The system stores this set of privileges in privilege masks. A privilege mask is a set of flag bits with one bit corresponding to each privilege. When a flag bit is set, the user acquires the corresponding privilege.

For each active job, RSTS/E keeps three masks:

- Authorized mask -- The set of privileges that the system manager gives to the account. You can use the SHOW ACCOUNT/FULL command to list the set of privileges available to your account.

- Current mask -- The set of privileges now in effect for the job. The system always references this mask when it performs a privilege check. You can raise or lower your privileges (up to your authorized limit) with the Set/Clear/Read Current Privileges SYS Call (SYS 28), or the DCL

SET JOB/PRIVILEGE command. You can list your current set of privileges with the SHOW JOB/PRIVILEGE command.

- Saved mask -- The saved record of the current privileges when a job gains temporary privileges (see the section "Temporary Privileges").

When a user attempts to perform an activity that is restricted by one or more privileges, the system performs a privilege check. This check examines the current mask to determine if the requesting job has all the privileges required to perform the activity. If the requesting job has insufficient privilege to perform the activity, the system returns one of the following errors:

```
?Protection violation (ERR=10)

?Illegal SYS() usage (ERR=18)
```

Multiple Privileges and Jobs

The following sections describe how the monitor handles privilege information during the life of a job. They describe:

- Job creation
- Login
- Logout
- Spawned jobs

Job Creation

At job creation, the monitor initializes both the current mask and the authorized mask, giving them all privileges except SYSMOD. This applies to all newly created jobs with the exception of those created by SYS 24, Create a Job (see Chapter 7).

Login

When a job logs in, the Login SYS call (SYS 4) looks up the authorized mask in the account attributes. It copies this mask into the saved and authorized masks, ORs it into the current mask, and sets the job status to indicate the job has temporary privileges in effect.

If a program logs in, it now has all the privileges it originally had, plus possibly some new ones. When a program exits, the user has all authorized privileges enabled.

A user who logs in may not want all his authorized privileges to be active at login. In that case the user can employ a LOGIN.COM file to initially drop some privileges.

Logout

When a job logs out, the monitor clears the group-related privileges GACNT, GREAD, and GWRITE in all three privilege masks. This is done because the job is currently running with PPN = 0, effectively putting it in group zero. The monitor drops group privileges because the intent of these privileges is to allow access to the user's group, not group zero.

Apart from losing group privileges, a job neither gains nor loses any privileges as a result of logging out. Note that the Logout SYS call (SYS 5) performs a self-kill except when the job currently has WACNT privilege.

Spawned Jobs

The Create A Job SYS call (SYS 24) creates a spawned job. For jobs spawned logged-in, the monitor usually gives the spawned job the same set of authorized and current privileges as the account it logs in to. This is done before the program, if any, is run. If the program is a privileged program, the usual additional privilege processing takes place (see the section "Running a Privileged Program").

As an option, the caller of the Create a Job SYS call can specify that the created job have fewer privileges.

Jobs spawned logged-out are given the same privileges as the job issuing the spawn function.

Spawning a job logged-in to an account other than the caller's requires accounting (GACNT/WACNT) privilege. Logged-out spawn requires WACNT privilege. Spawn therefore allows users with accounting privilege to create jobs that have some other account's privileges, possibly more than their own.

Writing Applications Using Multiple Privileges

When you write applications in RSTS/E V9.0, you must correctly use the multiple privileges features. The following sections explain how to best use multiple privileges within your program. They describe:

- Writing programs protected <124> and <104>
- Writing programs protected <232> (privileged programs)
- Performing access and privilege checks
- Program exit
- Multiple privilege system function calls

Writing Programs Protected <124> and <104>

Before V9.0, only a "privileged" user could run an executable program residing in a [1,*] account with a protection code of <124> (60+64). These programs could safely assume that anyone able to run the program had all the privileges required to perform all of the program's steps (an exception to this was POKE, which required the program to be run from account [1,1]).

In V9.0, the concept of "privileged" user is no longer all inclusive. If you have WREAD (world read) privilege, you can execute any program protected <124> on the system, even though you may not have all the privileges required for the program to work properly.

It may be acceptable to simply leave programs protected <124> as is. These programs will succeed or fail depending on the privileges of the user who executes them. However, some <124> programs may require the user to have several different privileges in order to succeed. If a user has some but not all of the privileges required, the program may partly succeed; it can complete some of its tasks but may fail at others. This may be undesirable, especially where failing part way through a multistep operation could leave a file or other data corrupted.

The solution to this problem is for such programs to do a privilege check at the beginning of the program, to ensure that the user has all the required privileges before proceeding. You can use the Check Access Function SYS call (SYS 32) to determine if a user has a particular privilege. See Chapter 7 for a complete description of this call.

Once you add a privilege check to <124> programs, you can safely lower the program's protection code to <104> (40+64). Protection code <104>

allows any user on the system to run the program. The up-front privilege check terminates the program if its user does not have the proper privileges.

For example, suppose a program requires HWCFG, SWCFG, and TUNE privilege in order to work properly. The program should initially perform a check to ensure that any user running the program has all three privileges before continuing. If the user has HWCFG and SWCFG privilege, but lacks TUNE privilege, then the program issues an error message and terminates.

If you still want program privacy, you can leave the program's protection code <124>, allowing only users with WREAD (or GREAD if the program resides in the same group as the user) to access the program or display it in a DIRECTORY listing.

Writing Programs Protected <232>

In some cases, you may not want to require users to have all the privileges that a program needs to work properly. In such cases, you can give a program temporary privilege by setting the privilege bit (128) in its protection code. When a privileged program is executed, it receives all privileges except SYSMOD.

Any program with a protection code of <192> or higher is privileged. The normal protection code associated with privileged executable programs is <232>, granting execute access to all, but restricting read/write access to the owner.

For security purposes, the system places two restrictions on privileged programs:

- You needs SYSIO privilege to designate a program as privileged.
- A privileged program that resides on a disk mounted /NOSHARE will not have temporary privileges when run. This restriction prevents an outsider from acquiring privileges by bringing in a privileged program on a private pack. To be able to mount a disk /SHARE, you need MOUNT privilege.

Privileged programs may be available to all users (for example, SYSTAT), or they may be restricted by including a check for some privilege at the beginning. Using the previous example, if you make a <104> program privileged (protection code <232>), it can check at the beginning for only TUNE privilege. The program proceeds for those users with TUNE privilege, even though the program itself requires HWCFG and SWCFG privilege as well. Be sure to drop temporary privilege before doing the privilege check, so that the user's

privileges are checked, not the program's (see the next section).

SHUTUP is an example of such a privileged program. It requires a variety of privileges to remove jobs, remove runtime systems, dismount disks, and issue the Shut Down System SYS call (SYS -16). Instead of requiring a user to have all of these privileges, SHUTUP is installed as a privileged program (protection code <232>) and only requires the user to have SHUTUP privilege in order to perform all of its steps. SHUTUP returns the error message ?SHUTUP privilege required if a user without SHUTUP privilege attempts to run it.

Whenever such a program drops temporary privilege, the program's privileges are saved and the user's own privileges are re-enabled. When temporary privileges are regained, the two sets of privileges are exchanged again. If temporary privileges are permanently dropped, then the user's privileges are re-enabled and the program's temporary privileges are lost.

You should be careful when you create privileged programs. In general, a privileged program should execute most of its functions with temporary privileges dropped, raising them just before executing a privileged operation and then dropping them immediately following the operation.

Pay special attention to BASIC-PLUS error handling under such conditions. If a privileged operation causes an error, control may be passed to an error handler with temporary privileges still enabled. Be sure that there are no paths in the program where temporary privileges may be accidentally left enabled.

Program Access and Privilege Checks

When designing programs, avoid duplicating the monitor's access and privilege checks in your program. When performing an operation that depends on the user's privileges and/or a file's protection code, a program should simply perform the operation (with temporary privileges disabled if a privileged program), and let the monitor enforce its access and privilege rules. Duplicating such checks in the program itself is inefficient and may lead to incompatibility in the future.

For example, suppose you want to design a privileged program that creates a file in a user-specified location (device and account). Rather than having the program determine if the user is authorized to create the file in the location specified, simply drop temporary privileges and create the file. If the user lacks the required privileges, the monitor blocks the file's creation and returns an error. The program can then report the error and reprompt the user for a new file location. Note that this program will continue to function properly, even if RSTS/E access and privilege rules change in the future.

Several system function calls allow programs to more easily establish access rights and privileges. DIGITAL recommends you use these calls where possible. See the section "Multiple Privilege System Function Calls" for a summary of the calls.

Program Exit

Whenever a program exits or chains to another program, the monitor performs the following privilege-related cleanup:

- If temporary privileges are in effect, the monitor cancels them.
- The monitor cancels any third-party privilege check currently in effect. (See the Third-Party Privilege Check SYS call, SYS 31)
- If the job is currently logged-out and does not have WACNT privilege, and the program exits, the monitor kills the job. Chaining among programs is possible without restriction when logged out, but other operations that exit the current program result in a self-kill. Note that the Logout SYS call (SYS 5) performs a self-kill immediately unless the caller has the WACNT privilege.
- If the program being exited is a privileged program, the monitor clears the job's memory and sets the job size to the minimum size for the job's default keyboard monitor.
- All open files are closed.

Multiple Privilege System Function Calls

Five SYS calls control multiple privileges:

- Drop/Regain Temporary Privileges (SYS -21) -- This call allows a program to selectively use temporary privileges.
- Set/Clear/Read Current Privileges (SYS 28) -- This call reads the current mask and selectively sets and/or clears bits in it. The SET JOB/PRIVILEGE and SHOW JOB/PRIVILEGE commands use this call.
- Third-Party Privilege Check (SYS 31) -- This call enables or disables third-party privilege checking. Server programs such as spoolers use this call to perform privilege checks for users who request the service.

- Check Access Function (SYS 32) -- This call performs a variety of privilege checking functions. It checks file access rights, converts a privilege mask to names, and converts privilege names to mask.

- Send Privileges (SYS 22) -- This new subfunction of the Send/Receive call permits a program to pass a job's current privileges to another program.

See Chapter 7 for a detailed description of each SYS call.

Non-File-Structured Disk Operation

Non-file-structured disk operation lets sufficiently privileged users (RDNFS, WRTNFS privileges) access specific blocks on a disk.

You can process non-RSTS/E file-structured disks under RSTS/E and use an entire disk as a single file. Non-file-structured processing also allows system programs, such as SAVE/RESTORE (see the *RSTS/E System Manager's Guide*), to optimally process file-structured disks.

Note

The data you look at when reading a disk as a non-file-structured device is internal to RSTS/E and is subject to change at any time.

Opening a Disk for Non-File-Structured Processing

If you have RDNFS privilege, you can open a disk in non-file-structured mode. To access a disk for non-file-structured processing, specify 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. An OPEN FOR OUTPUT statement results in the error ?Disk pack is not mounted (ERR=21). For example:

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

You need RDNFS privilege to read a disk that is open in non-file-structured mode. You need WRTNFS privilege to write to the disk. To prevent other programs from accessing a non-file-structured disk, a job with HWCTL privilege can assign the device.

Accessing Device Clusters

Before writing a program that accesses a disk as a non-file-structured device, you need to understand the terms logical block, device cluster, device cluster size, device cluster number, and default buffer size:

- A logical block is 512 bytes of disk data. Logical blocks are numbered starting at 0.
- 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 can 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 1-8.
- The default buffer size for all disk units when open in non-file-structured cluster mode is the device cluster size multiplied by 512 bytes.

Table 1-8 lists the default disk characteristics.

Table 1-8: Non-File-Structured Disk Default Characteristics

Device	Minimum Device Cluster Size	Default Buffer Size (Bytes)	Total Size (in Blocks)	Maximum Device Cluster Number
RX50	1	512	800	799
RK05	1	512	4800	4799
RK05F	1	512	4800 per unit; 2 units/drive	4799 per unit; 2 units/drive
RL01	1	512	10220	10219
RL02	1	512	20460	20459

Table 1-8: Non-File-Structured Disk Default Characteristics (Cont.)

Device	Minimum Device Cluster Size	Default Buffer Size (Bytes)	Total Size (in Blocks)	Maximum Device Cluster Number
RD51	1	512	21600	21599
RD52	1	512	60480	60479
RC25	1	512	50902 per unit 2 units/spindle	50901 per unit 2 units/spindle
RK06	1	512	27104	27103
RK07	1	512	53768	53767
RP02	2	1024	40000	19999
RP03	2	1024	80000	39999
RD53	4	2048	138668	34666
RM02/03	4	2048	131648	32911
RP04/05	4	2048	171796	42948
RM80	4	2048	242575	60643
RM05	8	4096	500352	62543
RP06	8	4096	340664	42582
RA60	8	4096	400175	50021
RA80	4	2048	237208	59301
RA81	16	8192	891056	55697
Virtual disk *	1	512	4 * #K words allocated	Varies with size

* The virtual disk is not a physical device. It is a logical device created from memory.

After you open a disk for non-file-structured processing, use the RECORD or BLOCK option in GET and PUT statements to read and write a

specific cluster on the disk. The number you specify designates a device cluster number. Thus, on an RK05, BLOCK 4100 refers to device cluster number 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 logical blocks. 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.

The system can access device cluster 0 only immediately after 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 COUNT to read a partial block (see the section "Partial Block Operations on Disk"), however the system positions itself at the start of the next cluster following the operation.

After you perform I/O to the disk, the only way you can access device cluster 0 is by closing the disk and reopening it for non-file-structured access. This statement reads the first block of an RK05:

```
100 OPEN "DK1:" AS FILE 1%
    \ GET #1%, BLOCK 0.
```

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. Writing to 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.

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 times the device cluster size. For example, the following statement opens

the RK05 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%
```

See the *BASIC-PLUS Language Manual* for a description of the RECORDSIZE option in OPEN statements.

Non-File-Structured Block Access: MODE 128%

Specify MODE 128% in a non-file-structured OPEN statement to access logical disk blocks instead of device clusters. MODE 128% lets you perform read/write operations on individual disk blocks.

To access blocks on the disk, specify MODE 128% in the OPEN statement and use the BLOCK option in the GET or PUT statement. The BLOCK option accepts a floating-point argument that represents the desired block (where block 1 is the first block on the disk, the pack label). See the *BASIC-PLUS Language Manual* for a description of the BLOCK option in GET and PUT statements.

Access to Bad Block Information: MODE 512%

MODE 512% in a non-file-structured OPEN statement allows a program to read beyond the last writable portion of a disk. The DCL INITIALIZE command uses this mode to read the factory bad block file, which is located beyond the last writable portion of the disk.

MODE 512% also suppresses errors normally logged by the system error logger. The system sends these errors to your program if you declare the program as a local receiver with object type code 64% (see Chapter 8).

Note that this mode is reserved for use by the disk initialization program and is not intended for general use.

Privilege and Access

You do not need to logically mount a disk that is being processed in non-file-structured mode. After you insert the disk into its drive, you can read or write to it if you have the appropriate privilege (RDNFS, WRTNFS). If you only have RDNFS privilege, you can read the disk regardless of the number of users accessing it, but if you attempt to write on the disk while another user is accessing it, a ?Protection violation error occurs.

If the disk is logically mounted, you have only read access while doing non-file-structured processing, unless you have both WRTNFS and SYSMOD privilege.

By testing bits 9 and 10 of the BASIC-PLUS variable STATUS, the user program can determine what accesses it has. See the *BASIC-PLUS Language Manual* for a description of the STATUS variable.

Allocating a Disk Unit

You can allocate a dismounted disk unit to your current job if you have the HWCTL privilege. This action prevents access by other users to the drive when you perform non-file-structured operations on a volume mounted in the drive.

When a dismounted disk is allocated, the system limits access to the drive. The drive cannot be logically mounted. If the job to which the drive is allocated has the necessary privileges, it has both read and write access to the disk. Other users who have the RDNFS or WRTNFS privilege can read the disk in non-file-structured mode but cannot write on the disk.

Allocating the disk unit can be useful when performing I/O. If you need to CLOSE and reopen and GET or PUT block 0, you do not lose ownership of the disk while it is closed.

The output of the SHOW DISK command shows an allocated drive as non-file-structured (NFS) and private (Pri). For example, the following portion of a SHOW DISK command output shows that disk DM1 is assigned.

```
Disk Structure:
Dsk Open     Size        Free     Clu  Err  Name    Level  Comments
DM1    1                            1    0                  Pri, R-O, NFS
DR1   45   131648    30052  22%    4    0   A       1.2    Pub, DLW, LDX
DR2    0   242576    33040  13%    8    0   R       1.1    Pri, R-O, DLW
DR3    8   500352    56296  11%    8    0   W       1.2    Pri, DLW, LDX
DR4    0   242572    17528   7%    4    0   M       1.1    Pri, DLW, LDX
DR5    0   500352    76152  15%    8    0   H       1.1    Pri, R-O, DLW
```

File-Structured Disk Operation

In file-structured disk operation, data is organized in files. The system manager uses the DSKINT option during system initialization or the DCL INITIALIZE command to set up a skeletal file structure on a RSTS/E disk. During timesharing, you can create files with the CREATE command, a text editor such as EDT, or the OPEN and OPEN FOR OUTPUT

statements. See the *BASIC-PLUS Language Manual* for a complete discussion of BASIC-PLUS I/O methods.

You can open disk files in one of several modes. The following sections describe these modes; Table 1-9 summarizes them.

Table 1-9: 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%	Random data caching (requires TUNE privilege)
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
8192%	OPEN file read only
16384%	Write UFD (requires WRTNFS privilege)

The general form of the OPEN statement with the MODE option is:

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

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

Note that if a nonprivileged job attempts to open a file in a mode that requires privilege, the system ignores that particular mode value. Table 1-9 lists the disk file MODE specifications.

Reading and Writing Disk Files: MODE 0%

Specify MODE 0% or omit the MODE option to open a disk file for normal reading and writing (the system default). 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 statements control only the actions the system performs when it opens the disk file. See the *BASIC-PLUS Language Manual* for a description of these statements.

Updating Disk Files: MODE 1%, MODE 4%+1%

In certain applications (for example, inventory updating) several users may need 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 gives several users 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.

RSTS/E File Updating Capabilities

In file updating operations, RSTS/E allows locks to be applied on blocks in 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.

The general form of the OPEN statement with the MODE option is:

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

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

Note that if a nonprivileged job attempts to open a file in a mode that requires privilege, the system ignores that particular mode value. Table 1-9 lists the disk file MODE specifications.

Table 1-9: 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%	Random data caching (requires TUNE privilege)
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
8192%	OPEN file read only
16384%	Write UFD (requires WRTNFS privilege)

Reading and Writing Disk Files: MODE 0%

Specify MODE 0% or omit the MODE option to open a disk file for normal reading and writing (the system default). 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 statements control only the actions the system performs when it opens the disk file. See the *BASIC-PLUS Language Manual* for a description of these statements.

Updating Disk Files: MODE 1%, MODE 4%+1%

In certain applications (for example, inventory updating) several users may need 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 gives several users 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.

RSTS/E File Updating Capabilities

In file updating operations, RSTS/E allows locks to be applied on blocks in 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.

File Update: MODE 1%

Use MODE 1% in the OPEN statement to open a file for update. For example:

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

This statement opens MASTER.DAT for update on channel 1 and creates a 512-byte buffer in your job space.

After a program opens 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 error ?Disk block is interlocked (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 exist for a program to unlock a locked block. The program can:

- Perform any write operation on the file.
- Execute the UNLOCK statement on the channel where the file is open. The UNLOCK statement has the form:

  UNLOCK <expression>

  where expression is the internal channel number of the file that is opened for update.
- Read another block. (However, this action locks the newly retrieved block.)
- Execute a CLOSE statement on the file. (Executing an END or CHAIN statement or executing the last statement of the program implicitly closes all files.)

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

You cannot open a file simultaneously in both normal and update mode. An attempt to perform an open in one mode when the file is currently open in the other mode generates the error ?Protection violation (ERR=10). The same error occurs if the protection code of the file prohibits read and write access.

Even if a file is open in update mode, a program can still gain read

access to the file. It can open the file with MODE 4096% (see the section "Reading a File During Processing: MODE 4096%"). This mode allows normal read access but not write access, regardless of whether the file is open 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. For example:

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

The RECORDSIZE 1024% option causes BASIC-PLUS to create a 1024-byte buffer. Therefore, a GET operation on channel 1 retrieves 2 blocks and puts both blocks together in the locked state. RSTS/E allows up to 31 blocks in the buffer to be locked in this manner and allows up to seven locks on the file (see the section "Disk Special Function: SPEC%"). Note that the same rules for a single locked block apply for the range of locked blocks.

You can open a file in UPDATE mode (1% or 5%) and extend it beyond the current end-of-file (EOF). To extend the file, follow these steps:

1. OPEN the file in UPDATE mode.
2. GET block 1 (the first block of the file).
3. Use the SPEC% function (see the section "Disk Special Function: SPEC%") to place an explicit lock on block 1.
4. Extend the file to the desired length beyond the current EOF with PUT statements.
5. Unlock block 1 (see the section "Disk Special Function: SPEC%").

The extended blocks are now available to users of the file.

Guarded File Update: MODE 4%+1%

Guarded file update in the OPEN statement provides the same update processing as MODE 1% with one more processing feature. The program can write a block or range of blocks only after it has read and locked the data. If your program attempts to write data that is not currently locked, the result is a ?Protection violation error (ERR=10). This feature prevents a program from updating data that it has not accessed. Note that you must use MODE 4% and 1% to gain special update; MODE 4% alone is equivalent to MODE 0%.

You can open a file in UPDATE mode and extend it beyond the current EOF. See the previous section for a description of the extend procedure.

Appending Data to Disk Files: MODE 2%

Use MODE 2% in the OPEN statement to write data to a new block following the current EOF in a disk file. Do not use the OPEN FOR OUTPUT statement, because it deletes the existing file. Specify MODE 2% only with block I/O files. For example:

```
100 OPEN "DATA.DAT" FOR INPUT 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 mode 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
```

Note that in certain applications, you may want to append records to a file on one channel and read the appended records on another channel. The most current file size information is available to all channels on which a file is open.

Special Mode for Extending Files: MODE 8%

Use MODE 8% in the OPEN, OPEN FOR INPUT, or OPEN FOR OUTPUT statement to force RSTS/E to update a file's size data and retrieval pointers on the disk during extend operations. In normal processing, RSTS/E maintains a file's size data in memory. RSTS/E does not update this size on disk until it allocates a new cluster to the file. By specifying MODE 8%, you force RSTS/E to update the on-disk file size as well as the retrieval pointers for each allocated cluster for every block added to the file. For example:

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

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

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 the system must correctly preserve a file's size in the event of a system crash or power failure.

Creating a Contiguous File: MODE 16%

Use MODE 16% 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. For example:

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

You can use other options with MODE 16% to specify the buffer size (RECORDSIZE) and the file cluster size (CLUSTERSIZE).

You must use the FILESIZE option with MODE 16%. It preextends the file to its maximum length, thereby telling the system how much contiguous space is required. If sufficient contiguous space is not available, the system generates the error ?No room for user on device (ERR=4). Note that you can specify MODE 64% (see the section "Creating a Contiguous File Conditionally: MODE 64%") to create a contiguous file conditionally. The file is made contiguous if possible; otherwise, it is made noncontiguous and no error is returned.

Processing a contiguous file greatly reduces overhead because it minimizes directory accesses and movement of read/write heads. Files for run-time systems and swapping must be contiguous because the monitor accesses these files independently of the normal file processor. However, you cannot extend a contiguous file. An attempt to extend a contiguous file generates the error ?Protection violation (ERR=10).

Creating a Tentative File: MODE 32%

Use MODE 32% in the OPEN FOR OUTPUT statement to create a file that does not become permanent until it is closed with the CLOSE statement. If a file of the same name currently exists, the system does not supersede it until you close the tentative file.

When you create a tentative file, the system searches for an existing file of the same name. If you do not specify an explicit disk name, the system searches the public structure. If the system finds a file of the same name, and its protection code does not allow deletion, you receive the error ?Protection violation (ERR=10). If the system finds a file of the same name, 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. If the system crashes or the job resets the channel (with a negative channel number in the CLOSE statement) before closing the file, the tentative file is deleted. Note that tentative file directory entries appear only 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, the error ?Protection violation (ERR=10) occurs. However, the system closes the tentative file and renames it to:

```
TM?nnn.TMP
```

where:

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

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

Creating a Contiguous File Conditionally: MODE 64%

Use MODE 64% in the OPEN FOR OUTPUT statement to create a conditionally contiguous file. MODE 64% causes the monitor to create a contiguous file based on the following conditions:

- If there is enough contiguous space available on the disk to contain the file, the monitor creates a contiguous file.

- If there is not enough contiguous space on the disk to contain the file, the monitor creates a noncontiguous file. If the monitor can create the file, it does not return an error.

Note that the monitor ignores MODE 64% if MODE 16% is also set for the file (see the section "Creating a Contiguous File: MODE 16%").

No Supersede: MODE 128%

Use MODE 128% in the OPEN FOR OUTPUT statement to create a file that 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. Instead, the system returns the error ?Name or account now exists (ERR=16).

Data Caching: MODES 256%, 2048%

When your job executes a read request, the monitor performs a disk access and transfers the requested data from the disk to the your job's I/O buffer. On systems with many jobs that use large amounts of data, the resulting large number of disk accesses can slow response time. You can reduce the number of data transfers from disk through data caching.

When you enable caching, the monitor stores the most recently read (accessed) data blocks in an area of memory called the cache, which is part of XBUF. If your 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 I/O buffer and thus avoids a physical disk access.

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

The system manager installs caching on the system and optionally sets its parameters during system start-up. When caching is enabled, the monitor examines the cache for all data transfer requests that are directed to the disk driver. If the requested data is in the cache, a read operation occurs without placing a load on the disk driver.

The monitor constantly updates the cache so that it contains the most recently requested data by adding data clusters or replacing data clusters (if the cache is full). The monitor schedules a job's data transfers into the cache based on the time since last access.

A data cluster currently in the cache is eligible for replacement if it:

- Is the data with the longest time since last access
- Has been in the cache for more than the minimum residency established by the system manager (the cache replacement timer, set with the SET CACHE command).

Cache Size

The amount of data that can be in the cache at any given time depends on the cache cluster size, which can be 1, 2, 4, or 8 blocks. In many cases, the cache cluster size determines the number of read requests that can be resolved in the cache before a disk access is required. For example, when the cache cluster size is 8 blocks, a read operation that installs data in the cache causes the installation of 8 physically contiguous blocks (including the requested blocks).

The system manager sets the cache cluster size during system start-up or with the Enable Disk Caching SYS call (SYS 19). For optimum performance, the cache cluster size should equal the pack cluster size set during disk initialization. If that is not possible, then the cache cluster size should be smaller than the pack cluster size. The monitor allocates cache space from XBUF (see the section "Enable/Disable Disk Caching," in Chapter 7).

Caching Control

If you have the TUNE privilege, you can enable or disable caching and determine the size of the cache by using the Enable Disk Cache SYS call (SYS 19) or the SET CACHE command (see the *RSTS/E System Manager's Guide*). In addition, if you have TUNE privilege, you can specify caching for a file on a system where caching is enabled.

You can cache a file in either random or sequential mode. Random mode is the default; DIGITAL recommends it for files that are accessed randomly, such as RMS indexed files. Sequential mode caching is designed for files that are accessed sequentially. If you are not sure in advance how a file will be accessed, you should specify random mode caching.

To specify caching for a file, you can either:

- Mark its UFD entry with the File Utility Functions SYS call (SYS -26) or the SET FILE command
- Specify MODE 256% or MODE 2048% in the OPEN statement

Both methods let you specify either random or sequential mode caching.

The best way to specify caching for a file depends on its use. If you are creating a file for use in a specific program, use the following MODE values to specify caching when you open the file. However, if you are creating a file for general use, it is better to mark the file's UFD entry with the File Utility Functions SYS call (SYS -26) or the SET FILE command. The use of caching MODE values requires TUNE privilege. However, a file whose UFD is marked for caching is cached on OPEN, regardless of the user's privilege, as long as caching is enabled on the system.

Random Mode Data Caching: MODE 256%

Use MODE 256% in the OPEN statement to cache data transfers to and from a file in random mode. MODE 256% has effect only if data caching is enabled on the system (see the section "Enable/Disable Disk Caching", in Chapter 7).

When a read on a randomly cached file occurs, the monitor examines the cache to determine if the requested data item is present. If the data is in the cache, the monitor copies the data from the cache buffer that contains it to the program's I/O buffer. The monitor then links the cache buffer to the beginning of the list of cache buffers and clears its time of residency since last access. The monitor maintains the list of cache buffers in order of increasing time since last access.

If the requested data item is not in the cache, the monitor examines the list of cache buffers to determine the time of last access for the oldest cluster in the cache. If the time is less than the minimum residency, the requested data cannot be installed in the cache, so the monitor automatically performs a normal disk read. If the time is greater than the minimum residency, the monitor replaces the current data in the cache buffer with the new data and then transfers it to the program's I/O buffer.

Sequential Mode Data Caching: MODE 2048%

Use MODE 2048% in the OPEN statement to cache data transfers to and from a file in sequential mode. MODE 2048% has effect only if the

file is being cached. That is, either MODE 256% is set, the file's UFD entry is marked for caching (see the section File Utility Functions, in Chapter 7), or caching is set for all data on the system (see the section "Enable/Disable Disk Caching", in Chapter 7). Note that sequential mode caching has no effect for a cache cluster size equal to 1, although no error is returned if the cluster size is 1.

Sequential mode works like random mode caching except for the way the monitor handles:

- A read on the last block of a cache cluster
- A read on more than one cache cluster

In sequential mode caching, a read on the last block of a cluster makes the cluster eligible for replacement, regardless of the amount of time it has been in the cache. This speeds the replacement process in the cache and minimizes the space that the cache requires. The monitor handles a read on any other block in the cache cluster the same as in random mode caching: the cluster becomes eligible for replacement only when its minimum residency time in the cache expires.

In a read on more than one cache cluster, the monitor transfers all the requested data blocks to the program's I/O buffer but only installs the last cache cluster in the cache. Furthermore, if the last data block read is the last block in a cache cluster, the monitor does not install any data in the cache. Thus, if you define the cache cluster size as 1 and specify sequential mode, no data blocks are installed in the cache because every data block is the last block in a cache cluster.

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

Use MODE 1024% to override the pack default and specifically place a file at the end of the current account's directory. This file placement is useful for 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 noncritical files at the end speeds access to the first part of the directory.

Use MODE 1024% only in the OPEN FOR OUTPUT statement to create a new file. If you do not specify MODE 1024%, the monitor places the file in the directory as directed by the pack default. This default depends on the system manager's response to the New files first? DSKINT question. 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 multidisk public structure (SY:), the monitor selects the disk pack with the most free space and uses that pack's default.

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

Specify MODE 1536% (MODE 1024% + 512%) in the OPEN FOR OUTPUT statement to cause the monitor to override the pack default and place a 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. This default depends on the system manager's response to the New files first? DSKINT question. 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 multidisk public structure (SY:), the monitor selects the disk pack with the most free space and uses that pack's default.

Use MODE 1536% for files that are frequently accessed. For example, if a program is used very heavily, you can place it at the start of the directory. For example, the $PIP program is heavily used on many RSTS/E systems. In this case, placing $PIP at the start of the [1,2] directory may improve system performance.

Reading a File During Processing: MODE 4096%

In certain applications, you may need to read a data file regardless of what other processing is in progress. Under normal circumstances, the system prohibits opening a file while the file is currently open for update (MODE 1% or MODE 4%+1%). However, with MODE 4096% you 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. For example:

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

You cannot perform write operations. If you attempt a write operation, the system generates the error ?Protection violation (ERR=10). If the file is simultaneously open for update, the system does not generate the normal error ?Disk block is interlocked (ERR=19) when the program reads a block being updated (although that block may contain inconsistent data).

Note

Use MODE 4096% with care because of the danger involved in reading data that is subject to change.

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 by 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 you do not want write access, use MODE 8192% in the OPEN FOR INPUT statement. The system never grants write access to a file opened with MODE 8192%.

You can use MODE 8192% on files that are opened normally (MODE 0%). However, you cannot use MODE 8192% to open a file that is currently opened for update (MODE 1%). If a file is currently opened for update, you must specify MODE 8192%+1% in order to open the file read-only. If the file is not yet opened and you specify MODE 8192%+1%, subsequent opens on that file must be made with MODE 1%. For example:

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

After execution of this 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 error ?Protection violation (ERR=10).

Write Access to a Directory: MODE 16384%

If you have the WRTNFS privilege, you can write into a directory by specifying MODE 16384% in the OPEN statement. For example, the following statement allows you to read and write into the UFD of account [5,10]:

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

An OPEN FOR OUTPUT statement is invalid for a UFD. Without MODE 16384%, the system allows only read access if you have the appropriate READ privilege (GREAD for group, WREAD for all).

Simultaneous Disk Access

RSTS/E permits several users to read from the same file simultaneously, but only one user can write to a file (unless the file is open in update mode). Without this limitation, two users could try to write the same record of the file simultaneously, 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. If a second user attempts to write into the file, the error ?Protection violation (ERR=10) results. Thus, users may fail to obtain write access 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. If your program attempts to do so, it results in the error ?Protection violation (ERR=10). However, a file can be open simultaneously in update mode and read during processing mode (see the section "Reading a File During Processing: MODE 4096%"). In addition, a file can 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 example in the section "Appending Data to Disk Files: MODE 2%", performs this check. See the *BASIC-PLUS Language Manual* for a description of the STATUS variable.

Disk Optimization

Whenever you open a file on the public structure, the system searches the directories of all public disks to determine whether the file exists. To avoid the overhead of searching multiple directories, you can put the file on a private disk.

When you dedicate a private disk to a large production file, it 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 the section "Creating and Placing a File at the Beginning of a Directory: MODE 1536%") to ensure better directory organization.

If you cannot do this, the system manager can optimally reorder the file directory with the REORDR system utility (see the *RSTS/E System Manager's Guide*). With REORDR, you can order files on an account in either forward or reverse direction, by either date and time of creation or date of last access.

When you create a large file, specify a large file cluster size to increase efficiency. A large cluster size reduces the number of UFD blocks required to describe the file. Performance improves because the system can read or write multiple blocks in a single transfer. In addition, you can preextend a disk file to its maximum length when you create it and can specify that contiguous space be used. Preextension 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.

If you have the appropriate accounting privilege (GACNT for group, WACNT for all), you can use the Create User Account SYS call (SYS 0) to optimally preextend and place directories. By doing this, you may improve system performance.

If you preextend a disk file with the FILESIZE modifier on the OPEN statement and you do not specify the cluster size with the CLUSTERSIZE modifier, the monitor computes the clustersize that is optimal for fast access. The monitor uses the formula FILESIZE/7, rounded up to the nearest cluster size. For example:

```
100 OPEN "MYFILE.DAT" FOR OUTPUT AS FILE 1%, FILESIZE 100%
```

This OPEN statement preextends the file MYFILE.DAT to a size of 100 blocks. The monitor automatically computes a cluster size of 16 (100/7, rounded up). Note that the largest possible cluster size is 256 blocks.

If a program requires simultaneous access to more than one data file, it is best to place 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.

Use different accounts to store different kinds of files. To minimize the number of poorly ordered accounts, dedicate certain accounts to files that are created once and remain fairly static, and reserve other accounts for transient files. 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.

Partial Block Operations on Disk

In general, the buffer you use for disk I/O should be a multiple of 512 bytes in length. Specify the buffer size by using the RECORDSIZE option in the OPEN statement.

By default, GET and PUT statements transfer the entire buffer. If you want to transfer less data, use the COUNT option. The COUNT option used in a GET statement specifies the maximum number of characters to be read in the current record regardless of the buffer size. In the following example the file is opened with RECORDSIZE 1024% and you want to read only 520 bytes:

```
100 OPEN "MYFILE.DAT" AS FILE 1%, RECORDSIZE 1024%
110 GET  1%, COUNT 520%
```

This 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 satisfy the COUNT of 520, 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 or PUT operations, you can use any value for RECORD or BLOCK. For example, with a COUNT of 520 bytes, BLOCK 1 accesses the first block and 8 bytes of the second block. BLOCK 2 in the GET statement retrieves the entire contents of the second block plus 8 bytes of the third block. The file is then positioned to access the block following the last one accessed (block 4 in the previous example).

For PUT operations, the COUNT must be a multiple of 512 bytes (or exactly 512 bytes when writing a UFD). For GET operations, COUNT must be even (a multiple of 4 on RP02/03 disks). In all cases, the COUNT value must not be greater than the buffer size (RECORDSIZE option of the OPEN). See the *BASIC-PLUS Language Manual* for more information.

The Virtual Disk - DV0:

The virtual disk lets you store temporary data within the system's memory. The virtual disk is not a physical hardware device, but it contains the same structures as a physical disk device. You can use the virtual disk for file-structured or non-file-structured I/O in the same way you use any other disk device, with one exception: all data written to the virtual disk is lost when the RSTS/E system shuts down or crashes. DV0: is the device designator for the virtual disk.

The system manager allocates memory to the virtual disk with the INIT.SYS DEFAULT option. Use the SHOW DISK command to find out if the virtual disk is enabled on your system.

You can use the virtual disk to store temporary files or any file that has a very short lifespan. Examples of temporary files are work files created by an application program like SORT/MERGE that are later deleted; virtual arrays created by BASIC-PLUS that are no longer needed once the program exits; or temporary files used for entering data in applications that give users a chance to edit data before updating a permanent file.

You can also place copies of read-only files that never change and are frequently accessed on the virtual disk. For example, place in virtual memory a copy of an index file that is used to access other files. Or, place heavily overlaid programs (like TKB) in virtual memory to improve performance.

The virtual disk is especially useful on large memory systems. Because the virtual disk never requires physical I/O, it is the fastest disk on your system. It is even faster than data caching for these reasons:

- A file placed on the virtual disk always remains in memory. On the other hand, a cached file remains in memory based on frequency of access.

- When you write to a file on the virtual disk, no physical I/O takes place. When you write to a cached file, physical I/O takes place. The file processor first performs a physical write, then it updates memory.

The virtual disk takes memory away from user space. On a small memory system, this may detract from overall performance. In addition, you cannot use the virtual disk for any permanent files because all data is lost when the system shuts down or crashes.

Asynchronous I/O Requests

An asynchronous read or write request performs the same basic function as the synchronous read or write request: it moves data between a device and a program. The difference lies in the completion of the request. While a synchronous request stalls the job's execution until the request is complete, an asynchronous request does not stall the program. The program continues to run regardless of the state of the I/O request. When the I/O request completes, the RSTS/E monitor executes an asynchronous completion routine (AST) in the user program. This routine notifies the user job of the I/O completion.

The AST is a section of code within the user job that executes when an I/O request completes. The AST is the only section of code in the program that can check for any device dependent errors.

BASIC-PLUS programmers cannot use asynchronous I/O. BASIC-PLUS-2 programmers can use this feature, but must write a MACRO subroutine. See the *RSTS/E System Directives Manual* for details.

Disk Special Function: SPEC%

The SPEC% function performs special operations on disks, flexible diskettes, magnetic tapes (see Chapter 2), line printers (see Chapter 3), terminals (see Chapter 4), and pseudo keyboards (see Chapter 4).

On disks, 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 the section Updating Disk Files). A locked range

(from 1 to 31 blocks) is one that cannot 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 format:

```
VALUE%=SPEC%(FUNCTION%, BLOCK, CHANNEL%, 0%)
```

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:

FUNCTION%=0%	releases all locked blocks.
FUNCTION%=1%	releases the current implicit lock.
FUNCTION%=2%	converts the current implicit lock to an explicit lock.
FUNCTION%=3%	releases 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%	converts 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.

0% is the handler index for disk devices.

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

```
100  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 error ?Disk block is interlocked (ERR=19). In addition, 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	ERR Value
?NO ROOM FOR USER ON DEVICE There are too many locks pending on this channel. 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 for FUNCTION% 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

RX01/02 Flexible Diskettes

The RSTS/E monitor handles the RX11/RX01 and RX211/RX02 flexible diskettes (sometimes called floppy disks) as non-file-structured devices. The device name for the flexible diskette is DX.

Note

The RX50 flexible disk is not in this category. It is treated as a file-structured disk with the device name DU.

BASIC-PLUS, which uses the standard monitor I/O services for flexible diskettes, lets you store only one file on a diskette. For example:

```
SAVE DX1:
```

This command stores one .BAS file on a diskette. To read the file from the diskette or to run it, use:

```
OLD DX1:
RUN DX1:
```

Two system utility programs, FIT and FLINT, let you store more than one file on a flexible diskette. These programs transfer specially formatted data between a flexible diskette and the RSTS/E environment. See the *RSTS/E Utilities Reference Manual* for more information.

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

Table 1-10 shows that you can open and access a flexible diskette in either of two modes.

Table 1-10: 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.

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
```

This interleaving algorithm is standard in other PDP-11 operating systems for the flexible diskette (for example, RSX-11M, RT-11). Note that track 0 is unavailable; its use is reserved for IBM-compatible labels.

The following statement opens the diskette on unit 3 in block mode on I/O channel 1:

```
10 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 you omit the RECORD option or include RECORD 0%, the next sequential block is read. For example:

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

where N% is the number of the sector at which the block begins. It can be any number from 1 through 493. (Only the first GET statement after the device is opened can access the first block on the diskette).

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

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

where:

N% is the number of the sector at which the block begins. The RECORD option can also include 16384% to write a Deleted Data Mark with each of the sectors (see the section "Deleted Data Marks").

C% must be a positive nonzero number.

You can perform block mode operations in sector mode. The following example opens an RX01 diskette with this statement:

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

Then use the GET (or PUT) statement:

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

where:

32767%+1%	specifies sector interleaving
N%*4%	defines 512-byte blocks at 4-sector intervals.

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:

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

When you use GET and PUT statements, you can calculate track and sector numbers from the RECORD number. If you specify the desired record number as N (any number from 0 through 2001), you can specify the track and sector to access as:

```
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 you omit the RECORD option or include RECORD 0%, the next sequential record is read. For example:

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

where N% is the record number and can be any number from 1 through 2001. (Only the first GET statement after the file has been opened can access record 0.)

If you include -32768% (formed by 32767% + 1%) in the RECORD option (for example, RECORD N%+32767%+1%), sectors are interleaved according to the algorithm discussed in the section Block Mode - MODE 0%.

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

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

where:

N% is the record number. The RECORD option can also include -32768% for interleaving (see the section Block Mode - MODE 0%) and 16384% to write a Deleted Data Mark (see the section "Deleted Data Marks") with each of the records.

C% must be a positive nonzero number.

Note

If you insert a single-density diskette into an RX02 drive, the buffer size 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 make sure that you read only one record, include COUNT 128% in the GET statement.

Flexible Diskette RECORD Modifiers

When you perform I/O operations on flexible diskettes, you can include three special RECORD values in GET and PUT statements to modify the actions of the diskette drive:

RECORD 8192%	Allows you to access logical record zero on the flexible diskette. Under normal operation, the system does not allow access to logical record zero after the first I/O operation is performed. However, the following statement accesses logical record zero: `GET #N%, RECORD 8192%`
RECORD 16384%	Writes a Deleted Data Mark to the diskette when used in the PUT statement (see the following section "Deleted Data Marks").
RECORD 32767%+1%	Causes the specified I/O operation to be performed in block mode. That is, when you want block mode 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.

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 error ?Data format error (ERR=50) occurs.

In 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. To write a Deleted Data Mark to a diskette, include RECORD 16384% in the PUT statement.

Partial Block Operations on Flexible Diskettes

Use the RECORDSIZE option in the OPEN statement on a flexible diskette to specify a value that is not a multiple of the default buffer size (512 bytes in block mode; 128 bytes or 256 bytes in sector mode). Be careful, however, in using the GET and PUT statements.

For GET operations with a nondefault 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 for RX01, times 2 for RX02); 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 three sectors, and the first eight 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.

You can use any legal value in the RECORD option with the GET statement. Thus, with a buffer size greater than 512 bytes, you can overlap record values to recover skipped data.

Note

> When you use the COUNT option in a GET statement, the COUNT argument must be a positive even number. If an odd number (or 0) appears in the COUNT, the error ?Illegal byte count for I/O (ERR=31) is returned.

For a PUT operation with a nondefault buffer size (or a multiple of

the default), the software performs the same skipping and positioning as with the GET statement. The software writes null bytes in the skipped data. If you include the COUNT option in the PUT statement, the software writes the specified number of bytes from the buffer and writes null bytes for the rest of the buffer and for the skipped data.

Flexible Diskette Special Function: SPEC%

The SPEC% function performs special operations on flexible diskettes, disks, magnetic tape (see Chapter 2), line printers (see Chapter 3), terminals (see Chapter 4), and pseudo keyboards (see Chapter 4).

For flexible diskettes, the SPEC% function lets you:

- Find out the density (single or double) of the current diskette
- Mount a new diskette and recompute the density
- Reformat an RX02 diskette for a desired density

Because the RX02 flexible diskette drive supports single- and double-density diskettes, the SPEC% function is 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. Normally 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 fail.

SPEC% permits you to include an instruction in your program that causes the driver to recompute the density. In addition, for RX02 flexible diskette drives, SPEC% lets you specify a density reformat operation.

The SPEC% function for flexible diskettes has the format:

```
VALUE%=SPEC%(FUNCTION%,PARAMETER,CHANNEL%,18%)
```

where:

VALUE% depends on the function code you specify in FUNCTION%.

FUNCTION% is a function code that specifies the desired operation. The codes are:

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 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.

18% is the handler index for flexible diskettes.

SPEC% can take up to 20 seconds to reformat the density of an RX02 diskette and cannot be interrupted with CTRL/C. If the operation is interrupted by power failure or catastrophic error, the diskette will contain both single- and double-density and cannot be used. To recover, you must reformat the diskette.

The following errors are possible during a SPEC% operation:

Meaning	ERR Value
?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% is useful in flexible diskette programming to make sure that sector opens are correctly handled. You can resolve the conflict between 128-byte single-density buffer sizes and 256-byte double-density buffer sizes by using the following procedure:

To field the buffer:

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

To write the buffer:

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

DENSITY% is defined as:

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

The Null Device - NL:

The null device exists as a debugging aid on all RSTS/E systems. 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 error ?End of file on device (ERR=11) and a write access simply returns control to your program.

You can use the null device to dynamically allocate buffer space in memory. It has a default buffer size of 2 bytes, which is adequate for performing alternate buffer I/O operations with data on another channel. To specify a different buffer size, use the RECORDSIZE option in the OPEN statement. The null device can use any even buffer size. For example, the following statement allocates 132 bytes of buffer space:

```
100 OPEN 'NL:' AS FILE 12%, RECORDSIZE 132%
```

Opening the null device is also a convenient way to set up a buffer for message send/receive operations. Use the RECORDSIZE option in the OPEN statement to specify the buffer size. See Chapter 8 for more information on message send/receive operations.

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

Chapter 2

Magnetic Tape

Magnetic tape is a compact, relatively inexpensive medium that can provide large amounts of off-line data storage. One reel of magnetic tape can store many files. In addition, through multivolume ANSI processing of the PIP system program, you can store one or more large files on several reels of tape.

Unlike disks, which can be accessed randomly or sequentially, magnetic tape is a sequential access device. In most applications, a magnetic tape file is read or written from beginning to end, and each record in the file is processed in order.

Magnetic tape is used for backing up disks on many RSTS/E systems. The RSTS/E BACKUP and SAVE/RESTORE programs (see the *RSTS/E System Manager's Guide*), the PIP program (see the *RSTS/E Utilities Reference Manual*), and the DCL COPY command (see the *RSTS/E System User's Guide*) can all perform this function. In addition, the RMSBCK and RMSRST utility programs (see the *RMS-11 User's Guide*) can back up and restore RMS-11 files between disk and magnetic tape.

Other uses for magnetic tape include journaling and data interchange. Some applications track transactions as they are processed by journaling each operation to a magnetic tape as well as to a disk. Magnetic tape is also useful for transferring data between different computer systems. Finally, you may want to use magnetic tape instead of disk for applications that require infrequent processing (particularly batch processing) and use large amounts of data.

Overview of Tape Operations

RSTS/E offers a variety of utility programs and software features for processing magnetic tapes. The utility programs can fill most general needs.

Magnetic Tape

This chapter discusses the software features, which provide extra flexibility and control for special applications. These features include:

- MODE values for use in file-structured and non-file-structured processing
- FILESIZE, CLUSTERSIZE, and POSITION values for ANSI tapes
- MAGTAPE and SPEC% functions

File-Structured and Non-File-Structured Processing

RSTS/E can process magnetic tape as either a file-structured or a non-file-structured device. File-structured processing lets you take advantage of built-in system file handling functions; thus, it is easier to program than non-file-structured processing. On the other hand, non-file-structured processing gives you more control over tape operations. (For example, you may need to process a tape written in a non-standard format by another system or recover a file from a corrupted tape in non-file-structured mode.)

Table 2-1 summarizes the BASIC-PLUS statements used to access magnetic tape on RSTS/E. These are the same statements used to access disks. See the *BASIC-PLUS Language Manual* for complete descriptions of the statements.

Table 2-1: Statements and Functions for Accessing Magnetic Tapes

Function	Stream ASCII (File-Structured)	Block I/O (File- or Non-File-Structured)
Open	OPEN	OPEN
Access Buffer	--	FIELD
Read	INPUT INPUT LINE	GET
Write	PRINT	PUT
Special	--	MAGTAPE, SPEC%
Close	CLOSE	CLOSE

The KILL and NAME AS statements (see the *BASIC-PLUS Language Manual*) apply only to disk and DECtape files; you cannot use them with magnetic tape files.

RSTS/E provides several MODE values for use with the OPEN statement to control file-structured and non-file-structured tape operations. The MODE values differ for file-structured and non-file-structured processing. The MAGTAPE and SPEC% functions, used mostly in non-file-structured processing, give you still more control over magnetic tape operations. In addition, the Special Magnetic Tape Directory Lookup SYS call (SYS 15) is available to look up directories on magnetic tape (see Chapter 7).

RSTS/E writes tape records of 512 bytes by default. Table 2-2 lists standard system defaults for magnetic tape density and parity. Note that all tape drives except for the TK25 use 9-track magnetic tape. The Set System Defaults SYS call (SYS 34) changes the system tape density default. See Chapter 7 for details.

Table 2-2: System Density Values for Magnetic Tape

Tape Drive	Density
TE10 TU10 TS03	800 bpi only
TE16 TU16 TU45 TU77	800 bpi 1600 bpi
TS11 TSV05 TU80	1600 bpi only
TK25 TK50	Special format

You can override the system defaults by using the MOUNT command. In addition, you can override both system and assigned defaults in a program by using the MODE option (in non-file-structured processing) and the MAGTAPE and SPEC% functions (in both file-structured and non-file-structured processing).

Magnetic Tape Labels

RSTS/E supports two types of magnetic tape file labels in file-structured processing: ANSI (American National Standards Institute) and DOS (Disk Operating System). These labels contain information about data on the tape, but they have different formats. The ANSI label has a more complex format and contains more information than the DOS label. A specific tape must contain only one type of label.

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 exchange. RSTS/E implements a subset of this standard.
>
> In addition, RSTS/E uses U (undefined) record format, which is not defined in ANSI standard X3.27-1978.

The system manager sets the default label format with the DCL SET SYSTEM command or with the Set System Defaults SYS call (34). If you want to use a different label, you can either select a label format for your current job with the MOUNT command or specify a label in a program by use of MODE values in the OPEN statement. The MOUNT command overrides the system default; the MODE values override both the system default and the job default.

Data and Label Handling in File-Structured Processing

File-structured magnetic tape processing involves two types of operations:

- Data handling
- Label handling

Data handling, which is done by your program, is no different from data handling on any other device: the operations you perform depend on the I/O method you use. In BASIC-PLUS, you can use either stream (formatted) ASCII or block I/O. Stream ASCII I/O limits you to stream ASCII records, but BASIC-PLUS takes care of record blocking and deblocking, buffer management, and conversion between ASCII and numeric data types. Block I/O lets you read or write any type of data record, but your program must do its own blocking and deblocking, buffer management, and data conversion. Note that you may be able to use PIP instead of writing your own program (see the *RSTS/E Utilities Reference Manual*). Or, you may be able to use the DCL COPY command (see the *RSTS/E System User's Guide*).

Label handling, on the other hand, is done by the system. (Your program needs to read and write magnetic tape labels only when you process tapes in non-file-structured mode.) The system needs information from you to write a tape label; you supply this information when you open the file. The way you supply information and the amount you supply depends on whether you are writing a DOS or ANSI tape.

In general, the system requires no special information from your program to write a DOS tape. You can use standard BASIC-PLUS programming techniques (such as the RECORDSIZE option in the OPEN statement to specify a buffer size other than the default). However, when you write an ANSI tape, you need to supply some special information, which you place in the CLUSTERSIZE and FILESIZE options and the POSITION switch when you open the file. CLUSTERSIZE, FILESIZE, and POSITION for ANSI tapes have different meanings than they do for disk files. These parameters:

- Specify information about record format and length to be written at certain positions in the tape label
- Determine the I/O buffer size
- Specify a section number for a multivolume file; that is, a file too large to fit on one tape

See the section "Processing ANSI Magnetic Tape Files" for more information.

Note that although the system writes the label based on information you specify, it does not check this information when you write data records to the tape. Instead, your program must ensure that the label information and the data format agree.

Reading a magnetic tape also differs depending on whether it has DOS or ANSI labels. When you open a DOS tape for input, the system creates a 512-byte I/O buffer unless you specify a different buffer size in the RECORDSIZE option. However, when you open an ANSI tape for input, the system determines the I/O buffer size from information in the label. Do not use the RECORDSIZE option when opening an ANSI tape.

The rest of this chapter describes magnetic tape operation in detail:

- File-structured processing
- Non-file-structured processing
- Multivolume ANSI processing
- MAGTAPE and SPEC% functions

- Asynchronous I/O processing
- Error Handling
- Programming Examples

Note that Appendix A of this manual describes DOS and ANSI label formats and explains how RSTS/E initializes the two types of tapes. This information is useful for reading a tape from another operating system or writing a tape for use on another operating system.

The File-Structured Magnetic Tape OPEN FOR INPUT

To open a magnetic tape file for file-structured processing, specify the device name and file name in the OPEN statement. For example:

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

The OPEN FOR INPUT statement searches for the specified file on a designated tape unit. Use OPEN FOR INPUT when you want to read a magnetic tape. Unlike disk operation, OPEN FOR INPUT on magnetic tape permits read access only. An attempt to write to the file generates the error ?Protection violation (ERR=10). If the system detects a logical end-of-tape before finding a file, the error ?Can't find file or account (ERR=5) occurs.

In the previous 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.

Table 2-3 shows the MODE values that you can use in an OPEN FOR INPUT statement. The MODE value can be the sum of any combination of these single values, as long as they do not represent conflicting operations.

Table 2-3: 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 system finds the file, it opens the file for read access only. If you later execute a GET statement on channel N%, it makes a block of the file available to the program in the channel's buffer.

For ANSI-labeled tapes, the system reads the block length from the header 2 label (HDR2) when it opens the file. The system creates the buffer at the size given by the block length. However, if the block length is odd, the system rounds the value down to make the buffer size an even number of bytes. (To avoid loss of data when a magnetic tape file is read, make sure the block length is an even value when you write the file.)

Under DOS file-structured operations, a GET statement reads magnetic tape records into a 512-byte buffer. However, in certain cases you may need to process records larger than 512 bytes. Use the RECORDSIZE option to allocate more buffer space than the default provides. The form of the statement is:

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

where:

N% is the internal I/O channel on which the file is open,

M% is the MODE value

R% is the desired record length. The system rounds R% down to an even number if R% is odd.

This statement opens the file FIDO under the current account on tape unit 0 for input and allocates R% bytes of buffer space for data transfer operations.

To open a file stored on a DOS file-structured magnetic tape under an account other than the current account, supply the project-programmer number in the OPEN statement. For example:

```
100 OPEN "[3,214]MT0:ABC" FOR INPUT AS FILE N%, MODE M%
```

In this example, the system associates tape unit 0 with the channel designated by N% and searches for file ABC under account [3,214] according to the value of M% in the MODE specification.

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 you changed the format when the unit was allocated to the job with the MOUNT command. If the label format differs or the tape is not properly positioned, the system generates the error ?Bad directory for device (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%.

Rewinding the Tape: MODES 2%, 32%, 64%

As mentioned before, MODE 0% reads the tape from its current position. If the file name 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 it either finds the file or detects the logical end-of-tape. 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 it finds the file or detects the logical end-of-tape. 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 you issue a CLOSE statement on channel N%.

Example of OPEN FOR INPUT Statement

You can use the MODE values in any combination as long as they do not represent conflicting operations. (For example, MODE 16384%+24576% causes illogical results because DOS and ANSI formats are mutually exclusive.)

Consider the following:

```
10 OPEN "MT1:NATHAN" FOR INPUT AS FILE 3%, MODE (32%+64%+24576%)
```

This statement opens the file MARKIE on tape unit 1 and associates it with channel 3%. You can also specify MODE 24772%, the sum of the three modes.

When the system executes this statement, it rewinds the tape to the first label record (MODE 32%) and begins to read successive file label records until it either finds the file or detects the logical end-of-tape. 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, INPUT, or INPUT LINE statements. Remember, since the file is open for input only, attempting to execute PUT or PRINT statements results in the error ?Protection violation (ERR=10).

The next CLOSE statement rewinds the tape (MODE 64%).

Reading Data

Three types of statements read magnetic tape data: INPUT, INPUT LINE, and GET statements.

If a tape contains stream ASCII data, you can read it with INPUT or INPUT LINE statements. These statements work the same way they do for disks.

To read other types of data, use the GET statement. GET reads a single block of data into the I/O buffer from a magnetic tape file that is open for input. Do not use both GET and INPUT statements to read the same file.

The GET statement for magnetic tape has the form:

```
100 GET #N%
```

where 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 you open the file. 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, you cannot use the RECORD option in the GET statement. After the GET, the number of bytes read is available in the RECOUNT variable. To associate string variables with all or part of the data in the I/O buffer, use a FIELD statement, (see the *BASIC-PLUS Language Manual*). Attempting to read beyond the end of the file results in the error ?End of file on device (ERR=11).

If the system reads a block that is larger than the buffer, it transfers the amount of data that fits, skips the excess data, and returns the error ?Magtape record length error (ERR=40). The next GET statement then reads the next block.

The GET statement does not perform any data conversions or record blocking and deblocking. Your program must interpret the data retrieved.

The File-Structured Magnetic Tape OPEN FOR OUTPUT

The OPEN FOR OUTPUT statement searches for a specified file on a designated tape unit. Use OPEN FOR OUTPUT when you want to write a magnetic tape. (Unlike disk operations, OPEN FOR OUTPUT on magnetic tape allows write access only.) 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 in the current account according to the value M% in the MODE specification. Note that the system ignores account numbers on ANSI-labeled tapes.

If it does not find the file, 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 or PRINT statement subsequently executed on channel N% writes the channel's buffer to the tape. Since the file is open solely for output, a GET, INPUT, or INPUT LINE statement executed on channel N% generates the error ?Protection violation (ERR=10).

The search is successful when the system locates the specified file. The value of M% in the MODE specification determines how the system searches for and acts on the file when it is found.

Table 2-4 shows the MODE values that can be used in an OPEN FOR OUTPUT statement. The MODE value can be the sum of any combination of these single values, as long as they do not represent conflicting operations.

Table 2-4: 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.

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 default unless you changed the format when the unit was allocated to the job using the MOUNT command.

If the label format differs or the tape is not correctly positioned, the system generates the error ?Bad directory for device (ERR=1).

If the system finds a file label record, and its file name (and account for DOS tapes) matches that of the file specified in the OPEN statement, the system generates the error ?Name or account now exists (ERR=16).

No match causes the system to rewind the tape and to check successive file label records until it either finds a match or detects the logical end-of-tape. 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%.

Writing a Label: MODES 16%, 512%

As mentioned before, a search is successful when the system finds the specified file on the magnetic tape. The error ?Name or account now exists occurs when this happens. This is a precaution to prevent you from unintentionally writing a file at this point. (Doing so will write over the current file and destroy all later files on the tape.) Include a value of 16% in the MODE specification to suppress this error message and cause the system to write over an existing file on magnetic tape.

Note

> Writing over a file causes any files after the overwritten file to be lost.

When 16% appears alone in the MODE specification, the system first reads the tape at its current position. If the system finds a file label record and the file specification in the label record matches the file specification 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 program write access to the file. If the logical end-of-tape is at the current position, the system backspaces one record, 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 it either locates the file or detects the logical end-of-tape. Detecting the logical end-of-tape before locating the file causes the system to backspace one record, write a tape label for the file, and allow write access to the file.

When you include 512% 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 the section "Rewinding the Tape"), takes precedence over 512%. Therefore, when you use 512% with any combination of values, not including 32%, the system writes a file record label at the current tape position.

Note

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

Extending a File: MODE 128%

When you include 128% in the value for the MODE option, the system attempts to open an existing file and position the tape so you can append information to it. The file must already exist; if it does not exist, the error ?Can't find file or account (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 the error ?Protection violation (ERR=10) occurs. As for all other MODE values, you can use 128% alone or with any combination of values.

DOS and ANSI Format Labels: MODES 16384%, 24576%

By default, the system assumes that label records on a tape (either DOS or ANSI) are in the system default format or the format you select for your job with the MOUNT command. The MODE values 16384% and 24576% override any current defaults for labeling.

MODE 16384% in the OPEN FOR OUTPUT statement causes the system to search for a specified magnetic tape file. The search succeeds only if the file is written in DOS format (that is, preceded by a DOS label).

MODE 24576% in the OPEN FOR OUTPUT statement causes the system to search for a specified magnetic tape file. In this case, the search succeeds only if the file label is written in ANSI format.

If the tape format (either ANSI or DOS) differs from that used in the search, the system generates the error ?Bad directory for device (ERR=1). If the system finds the file, it returns the error ?Name or account now exists (ERR=16).

The system reads the tape from its current position. If it does not find the file, the system rewinds the tape and reads file labels one by one until it finds the correct file. If the system detects the logical end-of-tape, it automatically backspaces over the logical end-of-tape, writes a DOS or ANSI label record for the file, and allows write access to the file.

Processing DOS Magnetic Tape Files

If the tape being processed is in DOS format, use the RECORDSIZE option in the OPEN FOR OUTPUT statement to designate the block length. Omitting the RECORDSIZE option from the OPEN FOR OUTPUT statement is the same as specifying RECORDSIZE 0. 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 the buffer size down to make it even.

To write blocks smaller than 512 bytes, create a buffer smaller than 512 bytes. Specify 32767%+1% plus an even value equal to or greater than 14 in the RECORDSIZE option. The minimum block for DOS format tapes is 14 bytes. For example:

```
100 OPEN 'MT1.ABC' FOR OUTPUT AS FILE 1%, RECORDSIZE 32767%+1%+130%
```

In this example, the 32767%+1% value sets the sign bit and tells BASIC-PLUS to use the value specified (130 in this case) instead of the default value of 512. If the sign bit is not set, the system creates a 512-byte buffer. If the value given is odd (and the sign bit is set), BASIC-PLUS rounds the buffer size down to make it even.

PUT statements write blocks on tape equal to the buffer size. You can use the COUNT option to write tape blocks smaller than the buffer size but not less than the minimum of 14 bytes.

Processing ANSI Magnetic Tape Files

If the system is processing a tape with ANSI labels, use the CLUSTERSIZE and FILESIZE options in the OPEN FOR OUTPUT statement to designate the record format and length, file characteristics, and block length. Use the /POSITION switch to specify a section number of a multivolume file.

The system uses these values to create the corresponding fields in the file label and to set the I/O buffer size. The FILESIZE and CLUSTERSIZE options and the /POSITION switch have effect only when the tape being processed has ANSI labels. The general form of the statement with options is:

```
10 OPEN 'MT0:ABC/PO[SITION]:n' FOR OUTPUT AS FILE N%,
   CLUSTERSIZE Q%, FILESIZE P%, MODE 24576% + M%
```

You must specify the options in the exact order shown; otherwise, the system generates the error ?Modifier error. To apply the system default for any option, omit that specification from its place in the statement.

In the previous example, 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. 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 succeed, the file name ABC must match the file identifier in the file label on the tape.

The value n in the /POSITION switch designates the section number of a multivolume file. If you do not specify the /POSITION switch, the default section number is 1. See the following section "Processing Multivolume ANSI Magnetic Tape Files."

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 2-5 shows the label data for values of Q%. The value specified with CLUSTERSIZE is the sum of values chosen from Table 2-5.

Table 2-5: 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% 4096% 8192%	M = carriage control embedded A = FORTRAN carriage control. (space) = Implied carriage control (when printed, line feed precedes and carriage return follows each record).

* RSTS/E undefined record format tapes cannot be processed directly by most other operating systems.
** RSTS/E does not support ANSI format S records.

If you omit the CLUSTERSIZE option from the OPEN FOR OUTPUT statement, the system applies CLUSTERSIZE 0%. The system creates a file with undefined (U) record format and embedded carriage control with record length 0%. (Use the default CLUSTERSIZE if you plan to use PRINT to write a stream ASCII tape.)

Note

U format records do not conform to ANSI standard X3.27-1978. Non-RSTS/E operating systems may not be able to read tapes with undefined format.

The record length that the CLUSTERSIZE option specifies is the value that 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 you use in the FIELD statement to subdivide the I/O buffer. The subdivisions created to load records into the I/O buffer then equal the record length on the 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 you omit the FILESIZE option (the same as specifying FILESIZE 0%) from the OPEN FOR OUTPUT statement, the system sets the block length to 512 bytes. In the FILESIZE option, you must specify a value between 18 (the minimum allowed on ANSI-labeled tape) and 4095. Because a record cannot span blocks, the FILESIZE value for fixed-length records must be a multiple of the CLUSTERSIZE value, and greater than the CLUSTERSIZE value for variable-length records.

In ANSI label processing, the system uses the block length from the HDR2 label to create the magnetic tape I/O buffer. This action allows 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.

You can use the FILESIZE option in ANSI label processing to create an I/O buffer other than 512 bytes. The specified block length is written in the HDR2 label. The block length on the tape should be an even number. If the block length is odd, the system rounds it down one byte to make the I/O buffer an even number of bytes.

Note that the action of the FILESIZE option in ANSI label processing is similar to the action of the RECORDSIZE option in DOS label processing. However, if you use the RECORDSIZE option in ANSI label processing, and the value you specify 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 advantage is gained from using a buffer size larger than the block length. Thus, DIGITAL recommends that you do not use the RECORDSIZE option in ANSI label processing.

Data to be written to ANSI-labeled tape is not automatically converted by RSTS/E to the appropriate ANSI record format. Your program must format the data in the I/O buffer before writing the buffer to the tape. In addition, 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 ANSI standard X3.27 - 1978. However, the PIP utility can create and read ANSI format records (see the *RSTS/E Utilities Reference Manual*).

Processing Multivolume ANSI Magnetic Tape Files

If you are processing large ANSI magnetic tape files, you can use the /POSITION switch in the file specification to label files that reside on more than one volume. The general form of the statement is:

```
10 OPEN "MTO:ABC/POSITION:n" [FOR OUTPUT/INPUT] AS FILE N%, MODE M%
```

where n indicates the volume number of the file. Legal values for n are:

OPEN FOR OUTPUT

0	Writes volume number 1 mark on the file
1-9999	Writes the volume number specified on the file. If you specify a value other than 0 or 1, the file must be the first data on the tape to ensure sequential processing.

OPEN FOR INPUT

0	Searches for the first file that matches the filename.ext
1-9999	Searches for the first file that matches both the file name, file type, and the volume number specified. If the file is found but the volume numbers do not match, the error ?Pack IDs don't match (ERR=20) is returned.

When you are at the the end of a tape and you know that there is more data for another tape, issue MAGTAPE function 10 (End-of-Volume Mark on CLOSE) before the CLOSE statement. When you issue the CLOSE statement, this MAGTAPE function writes an ANSI EOV label on the tape instead of the EOF label. See the section "The MAGTAPE Function" for more information on writing an EOV mark.

Multivolume magnetic tape processing works only on ANSI-labeled files.

Example of OPEN FOR OUTPUT Statement

You can use the MODE values available with OPEN FOR OUTPUT in any combination as long as they do not specify conflicting operations. For example:

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

This statement 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 the system executes line 10, it 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%); instead it continues to search for labels in DOS format from the next record on. If the correct label record is found (that is, 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. Since the file is open for output only, attempting to execute GET or INPUT statements results in the error ?Protection violation (ERR=10).

The next CLOSE statement rewinds the tape (MODE 64%).

Writing Data and Processing End-of-Tape

You can write data to a magnetic tape file with either PUT or PRINT statements. Do not use both statements to write the same file.

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 general form of the statement is:

```
100 PUT #N%
```

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

The PRINT statement writes stream ASCII data to a magnetic tape file. Use PRINT only if you plan to use the tape on a RSTS/E system. Other operating systems may not be able to read BASIC-PLUS stream ASCII data.

If RSTS/E finds the physical end-of-tape marker while writing to tape using a PUT statement, the system writes the entire record and returns the error ?No room for user on device (ERR=4).

However, if RSTS/E finds the physical end-of-tape marker while writing to tape using a PRINT statement, the system may not write the last item printed. The system 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. If you see this error, use one of these recovery procedures:

- Close the file as soon as the error occurs, and then create another file on another tape for the remainder of the data.

- If the tape is ANSI format and you want to use multivolume processing, follow these steps:

  1. Issue the SPEC% or MAGTAPE function to write an end-of-volume mark on the tape.

  2. Close the tape.

  3. Open the next volume of the file as the first file on another tape. Use the same name, but include the /POSITION switch to specify the next higher section number of the file.

  4. Continue writing the file on the next volume. If the error ?No room for user on device (ERR=4) occurs again, go to step 1.

- If the file is DOS format or if the file is ANSI format and you do not want to use multivolume ANSI processing, include a subroutine that writes a logical end-of-tape mark at the end of the previous file in the program. You can then write the file that generated the error condition to another tape. Follow these steps:

  1. Backspace with the MAGTAPE function using the maximum parameter 32767% (see the section "Backspace Function"). Repeat this procedure until the status function (see the section "Tape Status Function") indicates the tape is at beginning-of-tape (BOT) or that it detects a tape mark (end-of-file [EOF]).

2. If no error occurs during the backspace, check the tape status function (see the section "Tape Status Function") to see whether the tape is at BOT or EOF. If any error occurs, the data may be corrupt.

3. If the tape is at BOT, the file will not fit on the tape. Write three tape marks (see the section "Write Tape Mark Function") to zero the tape, then try a longer tape. Finding 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 finding BOT.

4. 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, and then write three tape marks.

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 you specify an OPEN FOR INPUT statement with M% in the MODE specification. An OPEN statement without a MODE specification is treated the same as MODE 0%. 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.

Use OPEN FOR INPUT or OPEN FOR OUTPUT instead of OPEN with magnetic tape. OPEN FOR INPUT and OPEN FOR OUTPUT allow the system to immediately determine which operation is needed.

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 EOF or EOV (if it is not already there) and frees the buffer space for other use 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 later output and frees the buffer space for other use within the program.

If you issue the Write EOV Mark on CLOSE MAGTAPE function (code 10) prior to the CLOSE, the system writes EOV labels instead of EOF labels.

In addition, the system rewinds the tape if you include the value 64% in the MODE specification when you open the tape. Otherwise, the system does not rewind the tape.

The Non-File-Structured Magnetic Tape OPEN

In non-file-structured processing, the system does no label processing. Essentially, the system passes all data directly between the magnetic tape and the user program. You can read or write tapes of any format with non-file-structured magnetic tape operations, as long as the program is set up to handle the actual tape format correctly. You can only write records of 14 bytes or longer. 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 error ?Illegal byte count for I/O (ERR=31).

To indicate non-file-structured processing, specify only the tape unit in the OPEN statement. Do not include a file name. There are three types of OPEN statements. The first two are:

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

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

The OPEN FOR INPUT and simple OPEN statements are equivalent. No tape movement occurs; the system permits both reading and writing of records.

The third form of the OPEN statement is slightly different:

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

In this example, the OPEN FOR OUTPUT statement permits writing only. The next section discusses this method of opening a tape for writing and the actions that occur on CLOSE.

The Non-File-Structured Magnetic Tape CLOSE

CLOSE has no special action on non-file-structured tapes unless you used an OPEN FOR OUTPUT statement. On a magnetic tape that is open for output, the CLOSE statement causes three trailing tape marks to be written, followed by backspacing over two of these tape marks, which positions the tape correctly for later output operations.

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

The MODE Specification in Non-File-Structured Processing

The MODE specification in non-file-structured magnetic tape processing can be used with some 9-track devices to indicate parity. For 800 bpi tape density, the standard parity is odd. DIGITAL does not recommend using the MODE specification to specify even parity. DIGITAL recommends the use of odd parity. Even parity, although available, cannot be used to write binary data. In addition, few other operating systems (or tape drives) support the use of even parity.

For 1600 bpi tape densities, parity is odd and nonselectable. The system ignores any attempt to specify even parity in the MODE specification.

See Table 2-2 for information on the density of 9-track devices.

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

```
D+P+S
```

where:

D (density) is:

```
   12 = 800 BPI
  256 = 1600 BPI
```

P (parity) is:

```
    0 = odd parity
    1 = even parity
```

S (stay) is:

```
    0 = MODE value does not stay after CLOSE
 8192 = MODE value stays after CLOSE
```

If you do not specify a MODE value in the OPEN statement, the system processes the tape using the system density default and odd parity.

If you add 8192% to the MODE value, the associated parity and density settings remain in effect for the job if the tape unit was allocated 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:

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

100 OPEN "MT0:" FOR INPUT AS FILE 1%, MODE 12%
```

Either statement makes the tape on the 9-track drive unit 0 available for execution of GET and PUT statements on channel 1%. The system accesses tape with a density of 800 bpi and odd parity. The system does not perform tape positioning or status checking. You must perform such operations using the MAGTAPE function described in the next section.

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

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

If the unit is write-locked (that is, the write-enable ring on the reel is removed), the system generates the error ?Device hung or write locked (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 error ?Protection violation (ERR=10). The system writes records in odd parity at a density of 800 bpi. Your program must check the status of the device and control the device by use of the MAGTAPE function described in the next section.

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

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

This statement associates the tape on unit 0 with channel 1%. The RECORDSIZE option creates a buffer of 1000 bytes. If insufficient memory is available, you see the error ?Maximum memory exceeded. You must then 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 one byte to make it even. If the number is less than 512, the system uses the default buffer length of 512.

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 error ?Magtape record length (ERR=40). The RECOUNT variable contains the number of bytes read.

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

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

This statement writes a 76-byte record. If you do not use COUNT, PUT writes an entire buffer, regardless of whether the buffer contains meaningful data. A record must be at least 14 bytes (18 bytes to conform to the ANSI standard), and no larger than the I/O buffer.

If a record smaller than the buffer size is read, the BASIC-PLUS RECOUNT variable contains the number of bytes read. Every input operation on any channel (including channel 0) sets RECOUNT. Thus, you should test or save RECOUNT immediately after each GET statement.

The MAGTAPE Function

The MAGTAPE function gives a program control over all magnetic tape operations. You can use MAGTAPE in either file-structured or non-file-structured processing, although it is mainly used in non-file-structured processing.

The general form of the MAGTAPE function is:

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

where:

- F% is the function code (1 to 12).
- P% is an integer parameter.
- U% is the internal channel number on which the selected tape is open.
- I% is the value returned by the function.

F% determines the effect of the MAGTAPE function. The following sections describe these functions, beginning with function code 1. In all examples in these sections, assume that tape unit 1 is open on channel 2.

Table 2-6 summarizes the MAGTAPE function codes and includes the designations IMMEDIATE and WAIT. IMMEDIATE means that the monitor starts the action and returns control to your program immediately; WAIT means that the monitor returns control to your program only after the operation is complete.

Table 2-6: MAGTAPE Function Summary

Action	Code	Parameter	Value Returned	Wait or Immediate
Rewind and offline	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	D+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
End-of-volume (EOV) labels on CLOSE	10	Unused	0	Immediate
Error condition acknowledged (only meaningful for asynchronous I/O)	11	Unused	0	Wait
Extended set density	12	Density to set/check	Actual Density set/checked	Immediate

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. For example:

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

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

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%)
```

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

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%)
```

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

Skip Record Function

```
Function code   = 4
Parameter       = number of records to skip (1 to 32767)
Value returned  = number of records not skipped (0 unless the system
                  finds a tape mark)
```

The Skip Record function advances the tape. The tape continues to advance until either the specified 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. (The system counts the tape mark as a record skipped.) For example, to skip from the current tape position to just past the next tape mark, use the function:

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

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

Backspace Function

```
Function code   = 5
Parameter       = number of records to backspace (1 to 32767)
Value returned  = number of records not backspaced (0 unless the
                  system finds a tape mark or BOT)
```

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 system counts the tape mark as a record actually backspaced.) The BOT is neither skipped nor counted as a skipped record. For example:

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

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

Set Density and Parity Function

Note

This function does not support the TK25, or TK50 magnetic tape drives. It is provided only for compatibility with existing software. DIGITAL recommends that the Extended Set Density Function (code 12) be used in future program development.

```
Function code   = 6
Parameter       = D+P+S
Value returned  = 0
```

where:

D (density) is:

```
   12 = 800 bpi
  256 = 1600 bpi
```

P (parity) is:

```
    0 = odd parity
    1 = even parity (not recommended; see the section "The
        MODE Specification in Non-File-Structured Processing")
```

S (stay) is:

```
    0 = MODE value does not stay after CLOSE
 8192 = MODE value stays after CLOSE
```

A tape drive is set to the system default for density and odd parity unless you change the default when you allocate the unit (with a MOUNT command) or when you open the unit. If the tape drive has more than one density and/or parity option available, this function changes the density and/or parity according to the value given as the parameter.

See Table 2-2 for information about 9-track tape drive densities, and the section "The MODE Specification in Non-File-Structured Processing" for information on parity settings.

The system interprets the parameter exactly as it does the MODE value in a non-file-structured OPEN statement. For example:

```
10          OPEN "MM0:" AS FILE 2%

20          I% = MAGTAPE(6%, 256%, 2%)
```

These statements set the density and parity of the 9-track tape drive open on channel 2 to 1600 bpi, odd parity. The density and parity

that you specify in the parameter are in effect until channel 2 is closed. The system sets I% to 0 to indicate successful completion. If you execute this function on a tape open in file-structured mode, the system ignores the request and returns the same value as the one passed.

If the unit is allocated, adding 8192% to the parameter value (making it 8192%+256%) keeps the new density/parity setting in effect even after the associated channel is closed. The next OPEN statement without a MODE option, associating any channel number with tape unit 0, automatically opens it with that new density/parity setting. A DISMOUNT command for a previously allocated unit returns 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, odd parity, using DOS labels. In this example, once channel 3 is closed, the new density/parity setting is now in effect and remains in effect until a DISMOUNT operation is executed on tape unit 2.

```
ASSIGN MM2:.DOS
OPEN "MM2:" AS FILE 3%
I% = MAGTAPE(6%, 8192%+12%, 3%)
CLOSE 3%
```

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.

Table 2-7 shows the status word format.

Table 2-7: 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 = reserved 1 = reserved 2 = reserved 3 = 800 bpi If bit 3 = 1, density: 0 = 1600 bpi 1 = reserved 2 = reserved 3 = reserved
12	(I% AND 4096%) = 0% (I% AND 4096%) <> 0%	9-track tape. Reserved.
11	(I% AND 2048%) = 0% (I% AND 2048%) <> 0%	Odd parity. Even parity.
10	(I% AND 1024%) <> 0%	Tape is physically write-locked.
9	(I% AND 512%) <> 0%	Tape is beyond physical EOT marker.
8	(I% AND 256%) <> 0%	Tape is at BOT (load point).
7	(I% AND 128%) <> 0%	Last command detected a tape mark (EOF marker).
6	(I% AND 64%) <> 0%	The last command was READ and the record read was longer than the I/O buffer size (that is, part of the record was lost).
5	(I% AND 32%) <> 0%	Unit is nonselectable (OFF-LINE).
4	(I% AND 16%) = 0% (I% AND 16%) = 1%	Unit does not accept 1600 bpi. Unit accepts 1600 bpi.
3	(I% AND 8%) = 0% (I% AND 8%) = 1%	See values for bits 14-13. See values for bits 14-13.

Table 2-7 (cont.)

Bit	Test	Meaning
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

Note

Bits 3, 4, and 11 to 14 are maintained only for backwards compatibilty. DIGITAL recommends that you use the Extended Set Density Function (code 12) for all future software development.

The following example obtains the status of the magnetic tape opened on internal channel number 2:

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

When the value of I% returned is 24,848 decimal (or 60420 octal), the magnetic tape is 800 bpi, 9-track, odd parity, and the last command issued was OFF-LINE. You can determine this information by testing the value of I%, bit by bit, against Table 2-7. For example:

```
I% = 24,848 (decimal)

   = 6 0 4 2 0 (octal)

   = 110 000 100 010 000 (binary)
```

The test for density uses bits 14 and 13:

```
(I% AND 24576%)/8192%
```

The following diagram shows the result:

```
          I% 110 000 100 010 000

AND 24576% 110 000 000 000 000

    Result 110 000 000 000 000
```

If you divide the result of (I% AND 24576%), which in this example is 24576%, by 8192%, the quotient can equal 0, 1, 2, or 3. In this case, 24576/8192 = 3, indicating that the tape density is 800 bpi.

The results of bit 12 (I% AND 4096%) and bit 11 (I% AND 2048%) are both zero, indicating a 9-track tape with odd parity.

Bit 8 (I% AND 256%) and bit 4 (I% AND 16%) both return a value of 1, indicating that the tape is at the load point and that the unit accepts 1600 bpi.

Bit 2-0 (I% AND 7%) returns a value of 0, indicating the last command issued was OFF-LINE.

Use the Skip Record function to advance to the next tape mark (that is, skip over the current file). You can use one Skip Record function unless the file is longer than 32,767 records (in which case the system must execute several skip record functions) or the system detects a physical EOT within a file. The following statements execute a Skip Record function until the next tape mark is found:

```
  20 I% = MAGTAPE(4%,32767%,2%)                          !Do some skips &
       \GOTO 20 UNLESS (MAGTAPE(7%,0%,2%) AND 128%)!Do more unless &
                                                         !tape mark found
```

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 file as a 16-bit integer, with certain bits set depending on the current file characteristics. Nonzero integers are returned for ANSI files; zero is always returned for DOS files.

Table 2-8 shows file characteristics word for ANSI format.

Table 2-8: 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 (embedded carriage control) 1 (FORTRAN carriage control) 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**

* ANSI format U does not conform to ANSI standard X3.27-1978
** RSTS/E does not support ANSI format S

The following example obtains 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 16464 (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. You can determine this information by testing the value of I%, bit by bit, against Table 2-8. For example:

```
I% = 16464 (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% (which in this case 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.

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%)
```

This statement rewinds the tape open on internal channel 2 when you issue CLOSE from a program or in immediate mode.

You must use the Rewind on CLOSE function after the OPEN statement and before the CLOSE statement. This function overrides all MODE specifications that, in the OPEN statement, instruct the system not to rewind on closing the file. Once the system executes the Rewind on CLOSE function, it cannot be cancelled.

Write End-of-Volume Labels on CLOSE Function

Function code = 10
Parameter = unused
Value returned = 0

This function writes end-of-volume (EOV) labels on the selected ANSI magnetic tape when the close statement is executed. This function is mainly for multivolume ANSI processing. For example:

```
I% = MAGTAPE(10%,0%,2%)
```

This statement causes EOV labels to be written to the file on execution of the CLOSE statement. Normally, end-of-file (EOF) labels are written. You must use the Write End-of-Volume Labels function after the OPEN statement and before the CLOSE statement.

This function works only on ANSI labeled magnetic tapes. An attempt to write end-of-volume labels on DOS-labeled or non-file-structured tapes results in the error ?Illegal MAGTAPE () usage (ERR=65).

Error Condition Acknowledged

Function code = 11
Parameter = unused
Value returned = 0

This function acknowledges an error condition that has occurred during an asynchronous I/O operation. When an error occurs while performing asynchronous I/O, the tape driver does not execute any more requests until this function has been issued. This is because asynchronous I/O allows multiple requests to be outstanding, but they may be invalid if the user knows of the error condition that occurred. All requests between the original errored request and the error condition acknowledged function call return the error ?Device hung or write locked (ERR=4). Once the error condition acknowledged function has been issued, the driver resumes normal processing, on the assumption that the user is aware of the error and is taking whatever steps are appropriate to correct it. For example:

```
I% = MAGTAPE(11%,0%,2%)
```

This statement acknowledges the error condition that occurred from the asynchronous I/O operation on the magnetic tape open on internal channel 2.

The Error Condition Acknowledged function returns no errors and will never fail when issued. If not required, it is simply ignored.

Extended Set Density Function

Function code = 12
Parameter = Density to set/check
Value returned = Actual density

You can use this function to set the density of a tape drive, or get density information about a drive. The action that RSTS/E takes depends on the value of the parameter.

If the parameter value is zero, RSTS/E returns the current density of the tape drive. If bit 15 is set, RSTS/E attempts to set the density of the tape drive to the value in bits 14-0 as follows:

Value	Meaning
32767	Sets the density to the highest legal density allowed for that tape drive. The value returned is the density set. No error is returned.
1	Sets the density to the lowest legal density allowed for that tape drive. The value returned is the density set. No error is returned.
n	Attempts to set the density to the value specified. If the value is not legal for that tape drive, RSTS/E returns an ?Illegal number error message (ERR=52) and leaves the density of the drive unchanged.

If bit 15 is clear, RSTS/E does not change the drive's density but only tests the value passed in bits 14-0 as follows:

Value	Meaning
32767	Returns the highest legal density for this drive
1	Returns the lowest legal density for this drive
n	Returns the nearest legal density for this drive that is not greater than the parameter value. If the parameter value is less than the drive's lowest legal density, RSTS/E returns the lowest possible density.

Any density changes made by this call, remain in effect until either a new density is set or a magnetic tape is read that has a density different than the one formerly set. This action is equivalent to the STAY value 8192% in the Set Density and Parity Function (function code 6).

Note

For MT, MM, and MU tape drives, a tape must be mounted and be at beginning-of-tape (BOT) to set the drive density. If this condition is not met and an attempt is made to change the drive's density, RSTS/E returns an ?Illegal MAGTAPE() usage error message (ERR=65). A tape does not have to be mounted on the drive to check legal densities or return the current density of a drive.

Asynchronous I/O Requests

An asynchronous read or write request performs the same basic function as the traditional synchronous read or write request: it moves data between a device and a program. The difference lies in the completion of the request. While a synchronous request stalls the job's execution until the request is complete, an asynchronous request does not stall the program. The program continues to run while the I/O request completes in the background.

When the asynchronous I/O request completes, the system informs the program that issued the request of the completion and status of the request. The system notifies the program by forcing it to run an asynchronous completion routine to notify the user job of the I/O completion. The asynchronous completion routine is a section of code within the user job that executes when an I/O request completes. When the asynchronous completion routine is entered, it can check for any device dependent errors.

Asynchronous I/O is only meaningful on MS: tapes (TS11, TK25, TSV05, TU80). Other tape drives accept asynchronous I/O requests and emulate asynchronous behavior, but the job stalls and few advantages are gained from its use.

BASIC-PLUS programmers cannot use asynchronous I/O. BASIC-PLUS-2 programmers can use this feature, but must do so using a MACRO subroutine. See the *RSTS/E System Directives Manual* for details.

Magnetic Tape Special Function: SPEC%

The SPEC% function performs special operations on magnetic tape, disks (see Chapter 1), flexible diskettes (see Chapter 1), line printers (see Chapter 3), terminals (see Chapter 4), and pseudo keyboards (see Chapter 4).

The SPEC% function for magnetic tape performs the same operations as the MAGTAPE function. It allows you to rewind the tape, skip records on the tape, and set tape density and parity. See the section "The MAGTAPE Function" for details.

The SPEC% function for magnetic tape has the format:

```
VALUE%=SPEC%(FUNCTION%,PARAMETER,CHANNEL%,14%)
```

where:

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

The code you specify in FUNCTION% determines the operation performed. These operations duplicate those performed by the MAGTAPE function codes (see Table 2-6). The following MAGTAPE and SPEC% functions are equivalent:

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

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

Magnetic Tape Error Handling

RSTS/E recognizes the following magnetic tape error conditions:

- Parity error
- Record length error
- Offline (not ready) error
- Write lock error
- Write beyond EOT error

For other error conditions that can occur with magnetic tape (Illegal byte count, File exists, Protection violation), see Appendix C.

Parity (Bad Tape) Error

If the system detects a parity error on a read attempt, it tries to reread the record up to 15 times. If the error condition persists, the error ?Data error on device (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 tries 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, the error ?Data error on device (ERR=13) occurs. In both cases, the tape is positioned just past the record on which the error occurred.

If you have error logging on your system, a magnetic tape error is logged for each parity error that occurs. Consult the ERRDIS full error report to see if the problem is due to a malfunctioning or poorly aligned magnetic tape drive.

Record Length Error

The record length 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, the error ?Data error on device (ERR=13) is returned to your program, and bit 6 of the tape status word is set. Therefore, if you are reading records of unknown length from magnetic tape, you must check for possible record length errors after every read operation. Use a statement of this form:

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

Note that if bit 6 is set in the tape status word, the IF condition in this example tests as TRUE. The error ?Magtape record length error (ERR=40) occurs when the tape block is too long, in either file-structured or non-file-structured magnetic tape.

Offline Error

The system determines the status of the tape unit by testing bit 5 of the returned value of the tape status function shown in Table 2-7. If bit 5 is set, the tape unit is offline. The error ?Magtape select error (ERR=39) occurs if you attempt to access an offline drive.

Write Lock Error

Attempting any write operation on a magnetic tape that is physically write-locked (that is, a tape that does not have the write-enable ring inserted) results in the error ?Device hung or write locked (ERR=14).

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; it does not harm the data. The program can recover in one of two ways; see the section "Writing Data and Processing End-of-Tape."

Magnetic Tape Programming Examples

The following examples show how to read and write a magnetic tape file.

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 "MM0: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 (MODE 32%) and when it executes a CLOSE statement on the file (MODE 64%). After the user types in each record, the program converts the data, builds a record, and writes the record to the file. Finally, after all records have been written, the program closes the file and ends.

Reading a Magnetic Tape File

The following BASIC-PLUS 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. For example:

```
  150  M%=16384%+64%+32%
       \OPEN "MM0: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 searches for the record the user specifies and converts the data in the record to a recognizable form before printing it.

Because magnetic tape is a sequential access device, the program uses the MAGTAPE function to backspace the tape to the beginning of the file following each record retrieval. This allows the user to request records in any order. Finally, the program closes the file and ends.

Reading a Magnetic Tape Non-File-Structured

The following program reads a DOS magnetic tape label record. See Appendix A for a description of the DOS label format.

```
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 file name (bytes 0-5) from Radix-50 notation to the ASCII character string F1$. The program then converts the project-programmer number (PPN) and protection code (P$, 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.

Chapter 3

Line Printer

RSTS/E provides several MODE and RECORD options as well as one SPEC% function for controlling line printer output. It also provides a FILESIZE modifier to enable extended software formatting. This chapter describes these options. In addition, it describes special character handling for line printers.

Special Character Handling

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

Table 3-1: LP11 Characters

Character	LP11 Action
CHR$(8)	BS - Backspace. This action depends on the /BACKSPACE qualifier of the SET PRINTER command. 1. Prints line 2. Returns carriage 3. Spaces to position immediately before previous position on line
CHR$(9)	Tab - Horizontal Tab. This action depends on the /TAB qualifier of the SET PRINTER command. 1. Spaces over to next tab position (columns 1, 9, 17, 25, and so on)
CHR$(10)	LF - Line Feed 1. Prints line 2. Returns carriage 3. Advances paper one line

Table 3-1: LP11 Characters (Cont.)

Character	LP11 Action
CHR$(11)	VT - Vertical Tab 1. Advances paper one line and resets line counter
CHR$(12)	FF - Form Feed 1. Prints line 2. Returns carriage 3. Advances paper to the top of the next form (see the section, Line Printer Control with the MODE Option)
CHR$(13)	CR - Carriage Return 1. Prints line 2. Returns carriage 3. No line feed (may be used for overprint)
CHR$(96) to CHR$(126)	Lowercase printing characters, converted to uppercase except on an uppercase/lowercase printer.

Line Printer Control with the MODE Option

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

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

The system associates line printer unit 0 with channel N%. The value of M% in the MODE specification determines the actions the system performs at the line printer.

Table 3-2 shows the line printer MODE values.

Table 3-2: Line Printer OPEN MODE Values

MODE Value	Line Printer Action
0% to 127%	Defines form length in number of lines per page. 0% indicates the default form length. You set the default form length with the SET PRINTER command. Also included when specifying nonstandard form length with software formatting (512%) and/or automatic page skip (2048%). This feature is maintained for backward compatibility only. Use the FILESIZE form (see the next section) in all new program development.
128%	Changes the character 0 (zero) to the letter O ("oh").
256%	Truncates lines that are longer than the form width. If MODE 256% is not set, then lines longer than the form width are wrapped onto the next line.
512%	Enables software formatting. Allows special characters to position paper at a specific line.
1024%	Translates lowercase characters to uppercase characters.
2048%	Skips six lines (that is, skips over perforation) at the bottom of each form.
4096%	Enables hardware form feed.
8192%	Suppresses form feed on CLOSE. Normally, two form feeds are generated whenever the line printer is closed.

Line Printer Control with the FILESIZE Statement

The FILESIZE specification in the OPEN statement allows you to use extended software formatting. This feature handles a line printer form length specification of up to 255 lines. It also enables two additional mode values: Change <ESC> to $ - MODE 16%, and Set NOWRAP - MODE 32%.

You enable extended software formatting with a FILESIZE 32767%+1% modifier in the OPEN statement. For example:

```
10 OPEN "LP:" AS FILE 1%, FILESIZE 32767%+1%+N%, MODE M%
```

The system associates line printer unit 0 with channel 1. The value N% specifies the form length and can be any value from 0-255. A value of 0 indicates the default form length. The FILESIZE value 32767%+1% sets the FILESIZE sign bit, thereby enabling extended use of the MODE values. M% specifies the MODE value.

Table 3-3 lists the MODE values available for use with the FILESIZE 32767%+1% modifier.

Table 3-3: Additional OPEN MODES with FILESIZE 32767%+1%

MODE Value	Line Printer Action
16%	Changes ESC to $. This mode disables escape sequences in data output to the device.
32%	Sets NOWRAP mode for lines that are longer than the printer's form width. Excess characters continue to be output to the device. Mode 256% overrides this mode.

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

Change ESC to $: MODE 16%

You can use MODE value 16% only when you include the FILESIZE 32767%+1% modifier in the OPEN statement. This mode value instructs the line printer driver to change any ESC character to a dollar sign ($) character. For example:

```
10 OPEN "LP:" AS FILE 1%, FILESIZE 32767%+1%+60%, MODE 16%
```

This statement enables extended software formatting and sets the page length to 60 lines per page. MODE 16% disables escape sequences in all data output to the device.

Set NOWRAP for Excess Lines: MODE 32%

You can use MODE value 32% only when you include the FILESIZE 32767%+1% modifier in the OPEN statement. This mode value instructs the line printer driver to continue to output excess characters to the device. For example:

```
10 OPEN "LP:" AS FILE 1%, FILESIZE 32767%+1%+60%, MODE 32%
```

This statement enables extended software formatting and sets the page length to 60 lines per page. The driver continues to output excess characters to the device.

Normally, the driver inserts a line feed character in lines that exceed the printer's form width, causing the line to be wrapped onto the next line. With MODE 32% enabled, the driver passes excess characters to the device without inserting a line feed character. the hardware characteristics of the device itself determine the actual display of excess characters. Note that the driver's horizontal position counter remains at the rightmost position of the form width, even though characters that exceed the line width are being sent to the device.

Note that MODE 256%, Truncate Long Lines, always takes precedence over MODE 32%.

Software Formatting: MODE 512%+N%

The MODE value 512% allows you to pass special control characters to position the paper on a specified line number. Note that if your system manager specifies 8 bit capabilities for a line printer (which allows 8 bit characters to be sent to the printer) you cannot perform software formatting to that printer. If you attempt to do so, the system generates the error ?Missing special feature (ERR=66).

For example:

```
100 OPEN "LP0:" AS FILE 1%, MODE 512%+30%
```

This statement enables software formatting and sets the form length to 30 lines per page. If you do not specify the form length, the system uses the default defined with the SET PRINTER command. Lines are numbered from zero to one less than the length specified. Thus, in the previous example, lines are numbered from 0 to 29.

After enabling software formatting with MODE 512%, you specify the line number on which to position the printer paper by sending a special character to the line printer in PUT or PRINT statements. The system skips to this line by sending the proper number of line feed characters to the printer.

The special character is of the form CHR$(128%+L%), where L% is the line number to advance to. For example:

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

This statement causes the system to advance the paper to line 19. If the line value L% is greater than the page length, the system ignores it. If the line value L% is greater than the current line number, the

printer skips to that line number on the current page. If the line value L% is less than or equal to the number of the current line, the system moves the paper to the top of the next page and then skips to the appropriate line.

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 aligned properly 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 you do not specify MODE 512% in the OPEN statement, and, if you do not specify 8 bit capabilities for the line printer, characters whose values lie in the range 128% to 255% are treated as (value - 128%).

Enable Hardware Form Feed: MODE 4096%

The form feed (FF) character advances the paper to the top of the next page. When you use the default form length, the FF character is sent directly to the device. If you use a form length other than the default, the system translates FF to the proper number of line feed (LF) characters to advance to the next page.

MODE 4096% causes the system to always send a FF to the device, regardless of the form length. This mode disables FF-to-LF translation. MODE 4096% is useful for devices that can be set to variable page lengths.

Note

> If you include both 4096% and 512% values in the MODE option, a FF character sent to the line printer remains untranslated. The form feed positions the paper at 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.

Translate Numeric 0 to Letter O: MODE 128%

A value of 128% in the MODE specification causes the system to print all 0 (zero) characters as O (uppercase "oh") characters. This feature is often used in commercial applications where there can be no possibility for confusion. For example:

```
10 OPEN "LP0:" AS FILE 1%, MODE 128%+60%
```

This statement indicates that the line printer should translate 0 to O (128%) on line printer unit 0 with a form length of 60.

Truncate Long Lines: MODE 256%

To truncate lines greater than the width of the line printer, include 256% in the MODE value. For example:

```
10 OPEN "LP0:" AS FILE 1%, MODE 256%+128%+22%
```

The statement sets the MODE value 128% on line printer unit 0; it also discards excess characters from each line printed (MODE 256%). The form length is 22 lines. When you do not use 256% in the MODE value, the system prints excess characters on a second physical line (unless you use MODE 32%).

Translate Lowercase to Uppercase: MODE 1024%

To translate lowercase characters to uppercase characters, include 1024% in the MODE value. For example:

```
10 OPEN "LP0:" AS FILE 1%, MODE 1024%+256%+128%
```

This statement sets the MODE values 128% and 256%. The default form length is used. In addition, it causes the system to translate all characters with representations between CHR$(96%) and CHR$(122%) to their equivalents between CHR$(65%) and CHR$(90%). The system also translates characters with representations between CHR$(224%) and CHR$(254%) to their equivalents between CHR$(192%) and CHR$(222%). This feature is always set for an uppercase-only printer.

Skip Lines at Perforation: MODE 2048%

To skip six lines at the bottom of each form, include 2048% in the MODE value. For example:

```
10 OPEN "LP0:" AS FILE 1%, MODE 2048%+1024%+256%+128%+60%
```

The statement sets the MODE values 128%, 256%, and 1024%, and also skips six lines at the bottom of each to page. Note that form length is specified by 60%. With MODE 2048% in effect, the system does not print on the last six lines of each form. This feature is useful when you are printing continuous listings to be placed in horizontal binders. If you load the line printer so that the top of form is the third physical line on the page, the system leaves three blank lines at the bottom and top of each page. When the listings are placed in binders, printed material is located three lines from the perforations of the page for easy reading.

Suppress 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 two form feeds (FF) on either an implicit CLOSE (for example, a CHAIN operation) or an explicit CLOSE. By specifying MODE 8192% in the OPEN statement, the program tells the system not to generate any form feed when it performs the CLOSE operation on the channel open for the line printer. For example:

```
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.

Line Printer Control with the 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 3-4 lists the values allowed in the RECORD option.

Table 3-4: 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 the system encounters an error.
8%	Clear pending output buffers before buffering characters for the request.
32%	Truncate long lines (enables MODE 256% for this output step).
4096%	Enable binary output, pass all characters to the device "as is."
8192%	Return control to the program if an output stall is to occur on the device.

The general format of the RECORD option for line printer operation is either one of these two forms:

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

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

The following sections describe the RECORD values.

Print Over Perforations: RECORD 2%

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

Delay Return Until Output Complete: RECORD 4%

For line printer output, the system transfers data from program buffers to the device by using intermediate storage areas called system 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 system buffers. At the same time, at its own speed, the line printer driver extracts the data from the system buffers and outputs it to the device.

Normally, completion of an output request occurs when the data is buffered. After buffering the data, the system returns control to the program at the next statement. If the program finishes its output routine but an error occurs at the device before the data is actually printed, recovery can be difficult under programmed control.

The RECORD 4% option in 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 -- although at the cost of increased execution time. To use this mechanism, print a NUL character with the RECORD 4% option. For example:

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

The output operation has no effect on the line printer because the system discards all NUL characters. The program maintains control of the output operation because the system does not complete the request until it prints all previously buffered characters. 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.

Clear Buffers Before Returning Control: RECORD 8%

Sometimes it is advantageous for a program to stop printing characters already buffered for output. Because characters to be printed on a line printer are kept in intermediate buffers, interrupting 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.

The RECORD 8% option in an output request tells the system to terminate the print operation and clear all pending output buffers before buffering the characters in the request. For example:

```
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 flushes out any characters in the printer hardware buffers by forcing them to print. After the successful completion of this statement, the printer and its buffers are clear, the vertical position counter is reset to top of form, and the horizontal position counter is reset to the left margin. (Although the driver's internal vertical form position counter is reset to top of form, you may need to align the form itself to its top-of-form position.)

Truncate Long Lines: RECORD 32%

RECORD 32% enables MODE 256% for one output step. RECORD 32% causes the driver to truncate lines greater than the width of the line printer.

Binary Output: RECORD 4096%

RECORD 4096% disables all formatting of characters sent to the line printer for one output step. The driver outputs all characters to the device "as is." Note that the driver does not update the vertical and horizontal position counters and the page counter when this modifier is in effect.

Note that you cannot output null characters to the printer when using binary output.

No Stall Option: RECORD 8192%

RECORD 8192% provides a "no stall" option for line printer output. RECORD 8192% causes the monitor to return control to your program if an output stall is to occur on the device. You can determine the number of bytes still to be written by checking the contents of the XRB+XRBC. The XRB is accessible only through MACRO; see the *RSTS/E System Directives Manual*.

RECORD 8192% is useful for programs that must perform several different functions with optimal performance (such as a line printer spooler that performs message send/receive and prints files at the same time). When an output stall does occur, the program can perform other processing before trying to write the remaining bytes to the line printer or terminal.

When you use the "no stall" option, you can perform a special test to see if the line printer is busy without causing your program to stall. To perform the test, print a single null character and specify RECORD (8192%+4%). When you specify both values, the system returns control

to your program instead of stalling it. If the system returns 0 at XRB+XRBC, the line printer buffers are empty, which means there are no characters still to print. A nonzero value at XRB+XRBC means that the line printer buffer still contains one or more characters to print. In this case, repeat the test until the system returns 0 at XRB+XRBC.

Note that BASIC-PLUS programmers cannot use this RECORD modifier. BASIC-PLUS-2 programmers can use this modifier, but must use a MACRO subroutine to check the XRB. See the *RSTS/E System Directives Manual* for details.

Line Printer Special Function: SPEC%

The SPEC% function performs special operations on line printers, terminals, disks, flexible diskettes, magnetic tapes, and pseudo keyboards.

For line printers, the SPEC% function lets you:

- Read the current value of the page counter.
- Read the current vertical and horizontal line positions.

The SPEC% function for line printers has the format:

```
VALUE% = SPEC%(FUNCTION%,PARAMETER%,CHANNEL%,6%)
```

where:

VALUE%	depends on the function code specified in FUNCTION%.
FUNCTION%	is the function code. The SPEC% function performs various functions on line printers as determined by the function code. These codes are: FUNCTION%=0 returns current value of page counter. FUNCTION%=1 returns current vertical and horizontal line positions.
PARAMETER%	is unused.
CHANNEL%	specifies the I/O channel for the line printer.
6%	is the handler index for line printers.

SPEC% subfunction 0 returns the current value of the page counter as a 16-bit value. SPEC% subfunction 1 returns a 16-bit value with the current vertical line position in the low byte and the horizontal position in the high byte.

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 unpredictable 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, continues output from the small buffers without loss of data. If a 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 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, your program must properly detect a line printer error condition and perform appropriate error handling. The system indicates a line printer error by generating the error ?Device hung or write locked (ERR=14). The first time the system returns this error after an output request (for example, 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. The monitor checks the printer's status every 10 seconds. It resumes printing when the error condition is removed. To prevent filling up monitor buffer space, subsequent output requests return immediately with the error ?Device hung or write locked (ERR=14). No data is buffered while the error condition persists. When an output request returns without error, the printer error is cleared. However, it is good programming practice to force the monitor to wait until line printer output is complete before printing any more data.

The following sample program demonstrates code that:

- Opens the line printer, inputs a line from the disk file, and performs output to the line printer
- Performs efficient error handling as described in this section

```
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

RSTS/E provides several features for use in interactive terminal applications. You access most of these features through the MODE option in the OPEN statement and the RECORD option in GET and PUT (or PRINT) statements. For example, by using various MODE and RECORD options you can:

- Display and process screen forms using echo control
- Perform I/O to several terminals using one I/O channel

This chapter describes these and other terminal features. It also describes:

- Escape sequences
- Private delimiters
- Pseudo keyboards

Except for the section on escape sequences, which contains information about the VT100- and VT200-family terminals, this chapter describes only the general-purpose software features that the RSTS/E operating system provides. See the user's guide for your terminal for hardware-specific information.

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. For example, 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 the section "Multiterminal Service on One I/O Channel"). Normally, the system stalls a program that is executing an input request until data is available in the keyboard input buffer (that is, until a user types a line terminator

at the keyboard). To avoid waiting for data, use RECORD 8192% in the GET statement. For example:

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

If data is available from the terminal open on channel 1, the system transfers it to the program's 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 error ?Data error on device (ERR=13). In both cases, the system reports the results immediately.

You can use RECORD 8192% with the SLEEP statement to wait for input. When you type a delimiter at a terminal or when a receiving job has received a message, the system cancels the sleep operation. This feature is useful for determining whether the sleep operation was canceled by terminal input or the expiration of a receive call's wait time (see the section "Receive" in Chapter 8). 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.

No Stall Option on Terminal Output: RECORD 8192%

When performing output to a terminal, you can also include the value 8192% in the RECORD option. Note that RECORD 8192% works differently for terminal input and output. When used on output, RECORD 8192% causes the monitor to return control to your program if an output stall occurs on the device. If an output stall does occur, the program can perform other processing before trying to write the remaining bytes to the terminal. This modifier performs a similar function to the "no stall" option for line printer output (see the section "No Stall Option" in Chapter 3).

Force Interactive Input: RECORD 256%

You can use the RECORD 256% modifier on a GET statement to force the program to always take input from the terminal, even if a command file is in effect. Normally, if you read from a terminal and there is a DCL command file active, then the program takes input from the command file. See the *RSTS/E Guide to Writing Command Procedures* for more information on DCL command files. For example:

```
GET #1%, RECORD 256%
```

This modifier is useful in programs that need to ask questions of a user, even when running under the control of a command file. The DCL command $INQUIRE uses this modifier.

Multiterminal Service on One I/O Channel: RECORD 32767%+1%

The multiterminal feature allows one program to interact with several terminals on one I/O channel instead of opening each terminal 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 control several terminals, you must first establish a master terminal by opening a keyboard on a nonzero channel. Two forms of the OPEN statement are possible:

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

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.

You can then control additional, or slave, terminals through special forms of the block I/O GET and PUT statements. The program must allocate the terminal to the job but must not open it. You can establish the terminals as slave terminals with the ALLOCATE command before you run the program. You can also allocate these terminals by executing the Allocate/Reallocate Device SYS call (SYS 10). Your program can control any number of terminals up to the maximum number of terminals on the system.

When a program interacts with several terminals on one I/O channel, the system services the terminals in round-robin fashion, determined by the numeric sequence of the terminals.

To perform input and output, use GET (or INPUT) and PUT (or PRINT and PRINT-USING) statements in a special manner, as the following sections describe. Note that the RECORD option specifies a particular action and keyboard number.

Multiterminal Service Output

Use a PUT statement of the following form to perform output to a keyboard, either master or slave:

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

where:

- K% is a variable in the RECORD modifier that specifies the unit number of the keyboard to which output is directed.
- N% is a variable in the COUNT modifier that specifies the number of characters to transfer from the buffer on channel 1 to the designated keyboard.

The only special error that can occur is ?Not a valid device (ERR=6), indicating that the terminal addressed is neither the master keyboard nor a slave keyboard reserved by the program. Other possible errors, such as ?I/O channel not open (ERR=9), work in the standard way.

You can use the RECORD option with the PRINT or PRINT-USING statement as well as with the PUT statement. For example, the following statements output the string Z$ to the unit designated by K%:

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

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

When you use PRINT or PRINT-USING, you do not need to use FIELD, LSET, and RSET statements to move data to an output buffer. It is also easier to format the data with PRINT or PRINT-USING than with block I/O statements.

You can output binary data using multiterminal service by including the value 4096% in the RECORD option. For example:

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

This statement outputs the number of bytes of binary data specified by M% to the keyboard whose unit number is the variable K%.

Note that when you use multiterminal service, the system keeps track of the current position (using the CPOS() function) of the output line of the master keyboard but does not keep track of the current position

of the output line of the slave keyboards. Thus, you should keep a count of characters printed to the slave keyboards if you need to know exactly what the current position is on the line.

Multiterminal Service Input

In multiterminal service, you can request:

- Input from a specific keyboard
- Input from any of the multiple terminals

You specify each type of input request by including certain values in the GET statement RECORD option. The rest of this section describes the two types of input requests in detail.

Use a GET statement of the following form to request input from a specific keyboard, either master or slave:

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

where the variable K% in the RECORD modifier specifies the keyboard number of the terminal from which input is requested. The GET statement transfers the data from the terminal's input buffer to the I/O 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 available in the RECOUNT variable. You can access the data with a standard FIELD statement. Because the first character of the I/O buffer is the keyboard number, the length of the data input is equal to RECOUNT-1%.

If no input is available from the designated terminal, the error ?Data error on device (ERR=13) results. Because this error is recoverable, your 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 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 system transfers the contents of that terminal's buffer 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 with input from a specific keyboard, you can use FIELD to access the sending keyboard number and the data sent. The variable S% tells the system how long to stall the program to wait for input. Table 4-1 lists the values S% can have.

If no input is pending from any terminal, the program stalls as described for 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 in the previous section. The error ?Data error on device (ERR=13) may occur due to a race condition with CTRL/C. No data is lost; simply reissue the GET statement to continue 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 cannot resolve these two 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 error ?Data error on device (ERR=13) occurs.
S% = 8192%	If no input is pending from any of the terminals, the error ?Data error on device (ERR=13) occurs immediately.

In multiterminal service, the system handles CTRL/C differently for slave and master terminals. A CTRL/C entered 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 entered at the master terminal, the system terminates the program in the standard fashion.

A CTRL/Z entered at either a master or slave terminal produces the error ?End of file on device (ERR=11). The system returns the unit number of the keyboard causing the error as the first character in the channel buffer.

Terminal Control with the MODE Option

You can control a terminal in several ways with the MODE option in the OPEN statement. Table 4-2 summarizes the MODE values you can use for terminals.

Table 4-2: Summary of MODE Values for Terminals

MODE	Meaning
1%	Enable binary input from a terminal
2%	Reserved for TECO
4%	Suppress automatic carriage return/line feed at right margin
8%	Enable echo control (turns off other modes and automatically enables MODE 4%)
16%	Guard program against CTRL/C interruption and dial-up line hibernation
32%	Enable incoming XON/XOFF processing
64%	Reserved
128%	Enable special scope RUBOUT
256%	Set escape sequence mode
16384%	Enable transparent control character output

The following sections describe the various MODE options.

Binary Data Output and Input: RECORD 4096% and 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, use a statement of the following form:

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

This statement transfers the number of bytes specified by M% to the output buffer of the terminal open on channel N%. You do not need any

special form of the OPEN FOR OUTPUT statement. Specifying RECORD 4096% in the PUT statement disables all output formatting on the terminal for that output operation.

You can obtain binary input from a keyboard by including MODE 1% in the OPEN statement. For example:

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

This 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.

A program can read binary data from:

- A terminal paper tape reader
- The terminal itself
- Any device connected to the system through a keyboard interface.

To start 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 program until data is received from the keyboard. When data is received, the system makes the program eligible to run and transfers the data to the program's I/O buffer. The program must execute GET statements often 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. You can override the default buffer size by using the RECORDSIZE option in the OPEN statement. However, because the system must first buffer the characters before they can be transferred to the program's buffer, changing the RECORDSIZE may not help increase the number of characters read by each read operation. (The system limit is approximately 180, but will vary depending on other system activity.) The RECOUNT variable contains the actual number of characters received.

Normally, the system terminates a read after every character typed at a terminal open for binary input. However, if you set one or more private delimiters for that terminal, the system terminates a read only when you type a private delimiter.

The system accepts and does not alter any characters received from a terminal open for binary input. Thus, entering CTRL/C has no effect. For this reason, the system disables binary input mode under any of the following conditions:

- The period for a WAIT statement expires. (The error ?Keyboard wait exhausted (ERR=15) occurs.)
- You execute any input or output statement on channel zero when the user's keyboard is open for binary input.
- You execute an OPEN statement in normal mode on the device but on a different channel.
- You execute a CLOSE statement on any channel associated with a keyboard open for binary input.

Under 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 error ?Keyboard wait exhausted (ERR=15). The keyboard stays open for normal ASCII data transfers.

Under condition 2, the system disables binary input mode when the program performs I/O on channel 0 and the user's keyboard is open for binary input on a nonzero 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 nonzero channel (channel 1). The statement at line 20 performs binary input from the keyboard. However, at line 40 the system executes a PRINT statement on channel 0, which disables binary input mode. The user's terminal remains open on channel 1 for normal ASCII data transfers. Note that a PRINT or PUT statement on channel 1 does not turn off binary input mode.

Under 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%
```

When the system executes line 100, it disables binary input on keyboard 6. If line 100 contained MODE 1%, the system would open keyboard 6 for binary input on channel 2. Therefore, keyboard 6 would be open for binary input on both channels.

Under 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%
```

The CLOSE statement at line 100 disassociates channel 2 from keyboard 6 but also disables binary input on channel 1. Keyboard 6 remains open in normal mode on channel 1. DIGITAL recommends using binary input mode by opening a device other than the user's terminal for binary input on any nonzero channel. Your program can interact normally with the user's terminal by executing standard INPUT and PRINT statements and can gather data from the binary device on the nonzero channel by executing GET statements.

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

Suppress Automatic Carriage Return/Line Feed: MODE 4%

RSTS/E normally performs a carriage return/line feed (CR/LF) operation when the right margin of a terminal is to be exceeded. (The SET TERMINAL command sets the right margin by means of the width characteristic.) You can suppress this automatic operation by opening the terminal with the MODE 4% option. For example:

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

The system opens keyboard number 13 on channel 1 in suppress CR/LF

mode. The system places all terminals allocated by the job but not opened in the same mode. (This action follows the multiterminal service rules; see the section, "Multiterminal Service on One I/O Channel.") Thus, all slave terminals have the same control characteristics as the master terminal.

MODE 4% stays in effect until the terminal is either closed or opened again without MODE 4%. All slave terminals stay in this 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, which the next section describes.

Echo Control: MODE 8%

RSTS/E normally echoes characters on your terminal as you type them. When you enter a BASIC-PLUS or DCL command, for example, the system displays the command on the screen. You can also "type ahead" on RSTS/E; that is, enter input faster than the system can process it. Besides storing your input in a type-ahead buffer to wait for processing, the system also echoes each character you type at the current cursor position on your screen.

While normal terminal echo is useful in many applications, it may be inconvenient in applications that display prompts or forms on the screen. For these applications, you can use echo control.

Echo control is designed for use on video terminals, but you can also use it on hard-copy terminals. It is often used with cursor control in forms-oriented data entry applications. Note that echo control is an optional feature of the RSTS/E monitor and may not be available on all systems.

Echo control modifies the way the system handles terminal echo. Instead of echoing characters as you type them, the system echoes characters only within a field that your program declares. If no field is currently active, the system stores typed characters and echoes them when your program declares a field.

Echo control mode also provides other features for screen applications:

- Automatic display of a "paint" character -- When you declare a field on the screen, you can define a special paint character for character deletion in the field. When you delete characters from the field, the system refreshes the paint character on the screen to maintain the appearance of the field.

The system maintains the declared paint character automatically; your program can display prompts or forms on the screen, accept input from one field at a time, and format the data for processing.

- Other special character handling -- For example, if you type too many characters in a field, the system can echo them as BEL characters or store them as input for the next field. You specify which type of processing you want when you declare the field.

To enable echo control, use the MODE 8% option in the OPEN statement:

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

where n designates the keyboard to be opened on channel 1 in echo control mode. A nonzero channel is required. The system also places all terminals allocated by the job but not opened in echo control mode. (This action follows the multiterminal service rules; see the section, "Multiterminal Service on One I/O Channel." Thus, all slave terminals are in the same mode as the master terminal.)

MODE 8% turns off other MODE options in effect (except MODE 16% and MODE 128%) and turns on MODE 4%.

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

- A CLOSE is performed on the channel
- The terminal is opened again without MODE 8%
- Any input or output is performed on channel 0 (the job's console terminal)

In echo control mode, the system strips the parity bit from all characters. All characters returned to the user have ASCII values in the range 1 to 127. The system does not pass synchronization and editing characters to the program. The system passes delimiters to the program but they are never echoed.

Table 4-3 summarizes how the system treats these characters in echo control mode.

Table 4-3: Echo Control Mode Character Set

ASCII Code	Code Returned to User	Comments
Ignored Characters		
0	--	Used as filler for timing.
Delimiter Characters		
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 (with line feed appended).
26	26	^Z (CTRL/Z combination); generates ERR=11.
27	27	If you use the SET TERMINAL/NOESCAPE_SEQUENCE command and output an escape character, the system returns 27 to the user and treats it as a delimiter. If you use the SET TERMINAL/ESCAPE_SEQUENCE command, the escape character triggers an escape sequence. The system returns an escape sequence to the user and considers the whole sequence as the delimiter.
125	27 or 125	If you use the SET TERMINAL/ALTMODE command, 125 is translated to escape (27). If you use the SET TERMINAL/NOALTMODE command, 125 is data.

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

ASCII Code	Code Returned to User	Comments
	Delimiter Characters	
126	27 or 126	If you use the SET TERMINAL/ALTMODE command, 126 is translated to escape (27). If you use the SET TERMINAL/NOALTMODE command, 126 is data.
	Editing Characters	
127	--	Rubout (DEL character); on video 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 Characters	
32-95	32-95	Normal 64-character graphic set.
96-126	64-94	If you use the SET TERMINAL/UPPERCASE=INPUT command, lowercase letters are translated to uppercase.
96-126	96-126	If you use the SET TERMINAL/LOWERCASE=INPUT command, lowercase letters are returned to the user.
192-254	192-221	If you use the SET TERMINAL/UPPERCASE=INPUT command, lowercase letters are translated to uppercase.

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

ASCII Code	Code Returned to User	Comments
	Data Characters	
192-254	223-254	If you use the SET TERMINAL/LOWERCASE=INPUT command, lowercase letters are returned to the user.
	Synchronization Characters	
17	--	XON (CTRL/Q combination); resumes suspended output (if you use the SET TERMINAL/TTSYNC command).
19	--	XOFF (CTRL/S combination); suspends output (if you use the SET TERMINAL/TTSYNC command).
	Other Characters	
1,2,5-9,11, 14-16,18,20 22-25,28-31	--	Echoed as BEL (code 7); otherwise, ignored.
17,19	--	If you use the SET TERMINAL/NOTTSYNC command, synchronization characters are treated as other (echoed as BEL; otherwise, ignored).

When you open the terminal in echo control mode, the terminal does no echoing until a field is declared on the channel. Declaring a field:

- Establishes field size, which is the maximum number of characters the field can hold.

- Specifies how overflow characters are handled. Two methods are available:

  - 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.

  - Keypunch. A field is terminated either by 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 a delimiter, a form feed (code 12) is appended to the field. The terminal does not echo any excess characters but retains them as input for the next field.

- 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. However, you can use a character like underscore (_) to indicate, or paint, the field. 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, execute a special form of the PUT or PRINT statement on the channel where the terminal is open with MODE 8%. Use the RECORD 256% and COUNT N% options in the PUT statement to declare the field:

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

where the value N% must be 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. Define the field as follows:

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 error ?Illegal byte count for I/O (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.

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.

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 for N% = 1%. The last byte is the paint character as described for N% = 2%.

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 you want to use a space as the 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.

You can also use the PRINT statement to declare a field, using a method similar to that of the PUT statement. The PRINT statement must include a RECORD 256% modifier to indicate the field declaration and string specifications (in place of the COUNT option) to declare field parameters. For example:

```
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 nonzero 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).

When you use the PRINT statement instead of the PUT statement to declare a field, it saves space in your program because it eliminates the need for the statements to define and load a buffer. Note that you should output all necessary bytes as one string, as in the previous examples. Do not use multiple elements separated by semicolons (;).

After you declare the field, the system enables echoing. The declaration makes the field active. The terminal then echoes characters until the field is filled. The terminal handles subsequent characters according to the overflow mode in effect for the field. When the terminal receives a delimiter (or the nth character for an n-character keypunch field), it deactivates the field and disables echoing. The terminal retains characters typed after the field is deactivated until the next field is declared.

Attempting to declare a field when one is currently active and the system has input characters for your program generates the error ?Account or device in use (ERR=3). The SYS call to cancel type ahead deactivates an active field.

You can combine 256% with other values in the RECORD option of the PUT or PRINT statement for multiterminal service operations. Combining RECORD values lets you declare a field for either the master or a slave terminal. You need not declare fields on all terminals, only on those terminals from which input is solicited. If your program tries to input data without declaring a field on any terminal, the system returns the error ?Data error on device (ERR=13).

DIGITAL recommends the following sequence when interacting with a video terminal in echo control mode:

1. Open any terminal on a nonzero channel with MODE 8%.

2. Execute the Cancel All Type Ahead SYS call (SYS 11), to cancel type ahead. This action discards any unsolicited input that 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 that will be maintained by terminal service during any deletion sequences.)

5. Position the cursor to the beginning of the first field (by direct cursor addressing).

6. Declare the field with the desired size and prompt and a paint character that matches 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. With the GET statement, you can use a 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.

For hard-copy terminals, the sequence is slightly different:

1. Open the terminal on a nonzero channel with MODE 8%.

2. Execute the Cancel All Type Ahead SYS call (SYS 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. Do not use INPUT, INPUT LINE, or MAT INPUT statements.

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 for each 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.

It is possible to use ODT submode (see SYS call 4, Enable ODT Submode) with echo control. Combining these features allows a program to examine every input character while ensuring that type ahead stays within the bounds of a field. However, some special processing is required for the program to work correctly.

Completion of an ODT submode input request does not necessarily terminate an echo control field. Therefore, if the program tries to declare the next field while the previous field is still active, one of two conditions occurs:

- If there is no pending input for the program, the system cancels the existing field and defines the new one.
- If there is pending input for the program, the system notifies the program by returning the error ?Account or device in use (ERR=3), and the field declaration fails.

To handle the second condition, you can trap the error and have the program read the rest of the characters in the field.

DIGITAL recommends using private delimiters instead of ODT submode (see the section, "Private Delimiters").

Prevent CTRL/C Interruption and Hibernation: MODE 16%

MODE 16% protects a program from:

- Aborting when CTRL/C is entered at the terminal.
- Hibernating when it becomes detached and attempts terminal I/O on a nonzero channel. A job becomes detached when it executes the detach system function call or when it is running over a dial-up line that gets hung up.

Entering 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%) causes a job to be detached and to enter the hibernation state as soon as it does terminal I/O. The job must wait until it is attached, through some external process, before it can recover. With MODE 16%, an immediate exit to the error ?I/O to detached keyboard (ERR=27) occurs when terminal I/O is attempted, which allows the program to recover. To take advantage of MODE 16%, your program must trap this error. Otherwise, the job goes into hibernation because BASIC-PLUS uses channel zero to display the error message on the terminal.

MODE 16% remains in effect until one of the following conditions is met:

- A CLOSE is performed on the channel
- The terminal is opened again, without MODE 16%
- Any I/O is performed on channel 0 (the job's console terminal)

Enable Incoming XON/XOFF Processing: MODE 32%

When an OPEN statement includes MODE 32%, an incoming XOFF character (ASCII 19) suspends output 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), the terminal processes all other incoming characters as for MODE 1%. However, the terminal ignores all other incoming characters when the OPEN statement does not also include MODE 1%.

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

- A CLOSE is performed on the channel
- The terminal is opened again without MODE 32%
- Any I/O is performed on channel 0 (the job's console)
- An input timeout occurs, producing the error ?Keyboard wait exhausted (ERR=15)

Special Use of RUBOUT: MODE 128%

MODE 128% allows video terminals to use RUBOUT as a delimiter. RUBOUT's use as a delimiter is subject to these conditions:

- If a typed character is the object of a RUBOUT operation and is a printing character (CHR$(32) to CHR$(126) or CHR$(160) to CHR$(254)), the terminal deletes the character.
- If there is no typed character or if the character is nonprinting (CHR$(0) to CHR$(31) and CHR$(127)), the terminal does not delete a character. The terminal buffers RUBOUT as a delimiter and marks the job as eligible to run.

The ability of MODE 128% to buffer RUBOUT as a delimiter is particularly useful to screen-oriented editors.

Note

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

Escape Sequence Mode: MODE 256%

When a terminal is in escape sequence mode, RSTS/E interprets the ESC character (CHR$(27%)) as the start of an escape sequence instead of as a delimiter. You can set escape sequence mode either by opening the terminal with MODE 256% or by setting the terminal's escape sequence characteristic with the SET TERMINAL/ESCAPE_SEQUENCE command. MODE 256%, the method DIGITAL recommends, sets escape sequence mode even if the terminal is set to /NOESCAPE_SEQUENCE.

DIGITAL recommends MODE 256% because, in addition to setting escape sequence mode, it modifies the way the system handles escape sequences that end with P. When you use MODE 256%, the system recognizes P as an escape sequence terminator. On the other hand, when you set the terminal's escape sequence characteristic, the system requires another character after P to terminate an escape sequence. See the section "Input Escape Sequences" for more information about escape sequence terminators.

Note

As an alternative to MODE 256%, an optional patch is available to cause the system to recognize P as an escape sequence terminator. Unlike MODE 256%, this patch affects all terminals on the system. See the *RSTS/E Maintenance Notebook* for details.

Because the system recognizes P as an escape sequence terminator with MODE 256%, you can use the same code to read incoming escape sequences from all keys on VT52-, VT100-, and VT200-family terminals in ANSI mode. On the other hand, when you set escape sequence mode through the terminal's escape sequence characteristic, you cannot use the same code to read incoming escape sequences from the VT100- or VT200-family PF1 key and the VT52 blue key, which are in the same place on the keypad. Both keys send escape sequences that end with P. See the section "Escape Sequences" for a complete description of escape sequences.

The following statement opens keyboard unit 46 in escape sequence mode on I/O channel 2:

```
100  OPEN "KB46:" AS FILE #2%, MODE 256%
```

This mode follows multiterminal service rules, which means that all terminals allocated but not opened by the job are also placed in escape sequence mode. See the section "Multiterminal Service on One I/O Channel" for more information about multiterminal service.

MODE 256% remains in effect until either:

- A CLOSE is performed on the channel
- The terminal is opened again without MODE 256%

If the terminal's escape sequence characteristic is set, escape sequence mode stays in effect when you cancel MODE 256%.

Escape Sequences

An escape sequence is a series of characters that performs a control function on the terminal, such as moving the cursor forward or backward or erasing part of the screen. The first character of an escape sequence is an ESC. The ESC character is a prefix that causes the terminal to treat subsequent characters as a command instead of echoing them on the screen.

One common use of escape sequences is cursor control. Cursor control is a feature of many video terminals, including the VT100-family and VT200-family terminals. As its name suggests, cursor control allows a program to manipulate the screen cursor. Cursor control is often used with the RSTS/E echo control feature in data entry applications.

This section:

- Summarizes commonly-used escape sequences for the VT100-family and VT200-family terminals.
- Shows how to use ANSI-compatible escape sequences to control the cursor and use two graphics features: reverse video and double-height characters. (The example is intended to show the technique, not to be a practical application.)
- Explains how the system handles input and output escape sequences for all types of terminals.

VT100- and VT200-Family Escape Sequences

VT100- and VT200-family terminals can operate in either ANSI-compatible mode or VT52-compatible mode. Each mode has a different set of escape sequences.

In VT52-compatible mode, the VT100- and VT200-family terminal responds to escape sequences like a VT52 terminal. VT52-compatible escape sequences let you execute programs on the VT100- and VT200-family terminals that are written for the VT52 terminal but do not let you take advantage of advanced features, such as reverse video. In addition, VT52-compatible escape sequences are not ANSI-standard.

If you write programs for both the VT52 and the VT100- and VT200-family terminals or if you are converting from the VT52 to a more advanced terminal, be aware of differences between the terminals. For example:

- The "home" cursor position differs for the VT100- and VT200-family terminals and the VT52. Home, which is the top left corner of the screen, is:
  - (1,1) for the VT100- and VT200-family terminals in ANSI-compatible mode
  - (32,32) for the VT52 and the VT100- and VT200-family in VT52-compatible mode

- When you use cursor control functions on the VT52 or the VT100- and VT200-family terminal in VT52-compatible mode, you must output the line and column positions as one-byte ASCII values. (You can use the CHR$ function to perform the necessary conversion.)

  On the other hand, when you use cursor control functions on the VT100- and VT200-family terminals in ANSI-compatible mode, you output line and column positions as string data. No conversion is necessary.

See the appropriate hardware manuals for a complete discussion of terminal hardware and software. The rest of this section describes VT100-family and VT200-family ANSI-compatible escape sequences.

Table 4-4 summarizes the VT100- and VT200-family ANSI-compatible escape sequences that move the cursor, erase all or part of the screen, and control line size and VT100- and VT200-family character attributes (bold, underscore, blink, and reverse video). The table uses the symbols Pl, Pc, and Pn:

Pl means line number.

Pc means column number.

Pn is a decimal parameter expressed as a string of ASCII digits. The parameter's meaning for each escape sequence is explained in Table 4-4. Separate multiple parameters with a semicolon (;). If you omit a parameter or specify 0, the terminal uses the default parameter value for that escape sequence.

Be sure to include the left square bracket ([) in the escape sequence prefix where Table 4-4 indicates. Note that escape sequences cannot contain embedded spaces. See the *VT100 User Guide* for a complete description of VT100 escape sequences. See the *VT220 User Guide*, *VT240 User Guide*, or *VT241 User Guide*, for a complete description of VT200-family escape sequences.

Table 4-4: ANSI-Compatible Escape Sequences: VT100- and VT200-Family Terminals

Escape Sequence	Description
	Cursor Movement
ESC[PnA	Moves the cursor up n lines without affecting the column position. The parameter Pn specifies the number of lines. The default value is one line.
ESC[PnB	Moves the cursor down n lines without affecting the column position. The parameter Pn specifies the number of lines. The default value is one line.
ESC[PnC	Moves the cursor forward (right) n columns without affecting the line position. The parameter Pn specifies the number of columns. The default value is one column.
ESC[PnD	Moves the cursor backward (left) n columns without affecting the line position. The parameter Pn specifies the number of columns. The default value is one column.
ESC[Pl;PcH	Direct cursor address. Moves the cursor to the specified line and column position. If you do not specify a line or column position, the cursor moves to the home position, which is the top left corner of the screen.

Table 4-4: ANSI-Compatible Escape Sequences: VT100- and VT200-Family Terminals (Cont.)

Escape Sequence	Description
	Cursor Movement
ESCD	Index. Moves the cursor to the current column position on the next line.
ESCM	Reverse index. Moves the cursor to the current column position on the preceding line.
ESCE	Moves the cursor to the first column position on the next line.
	Erasing
ESC[K or ESC[0K	Erases from the current cursor position to the end of the line.
ESC[1K	Erases from the beginning of the current line to the cursor.
ESC[2K	Erases the entire line containing the cursor.
ESC[J or ESC[0J	Erases from the current cursor position to the end of the screen.
ESC[1J	Erases from the beginning of the screen to the current cursor position.
ESC[2J	Erases the entire screen.
	Line Size (Double Height and Double Width)
ESC#3	Changes the current line to the top half of a double-height, double-width line.
ESC#4	Changes the current line to the bottom half of a double-height, double-width line.
ESC#5	Changes the current line to a single-width, single-height line.

Table 4-4: ANSI-Compatible Escape Sequences: VT100- and VT200-Family Terminals (Cont.)

Escape Sequence	Description
Line Size (Double Height and Double Width)	
ESC#6	Changes the current line to a double-width, single-height line.
To display double-height characters, use the ESC#3 and ESC#4 sequences as a pair on adjacent lines and send the same characters to both lines. The use of double-width characters reduces the number of characters on each line by half.	
Character Attributes (Require Advanced Video Option on VT100)	
ESC[Pn;Pn;Pn;...;m	Turns bold, underscore, blink, and reverse video attributes ON and OFF. Pn can have the following values: 0 or none All attributes OFF 1 Bold ON 4 Underscore ON 5 Blink ON 7 Reverse video ON The terminal executes the parameters in order and ignores any other parameter values. Unlike line size commands, which affect only the current line, the character attributes affect the entire screen. Remember to turn them OFF before ending your program.

Programming Example

The following example shows how to use VT100- and VT200-family ANSI-compatible escape sequences in BASIC-PLUS. The program uses PRINT statements to send the escape sequences to the terminal and uses the special value CHR$(155%) for the ESC character. (See the section "Output Escape Sequences.") Each PRINT statement ends with a semicolon to prevent BASIC-PLUS from printing a carriage return/line feed (CR/LF) as the last step in the PRINT statement. You need separate PRINT statements to print each half of the double height line.

```
10  EXTEND
100 ESC$ = CHR$(155%)                   !Set up variables
120 PREFIX$ = ESC$ + '['                !for ESC and ESC[ prefix
125 CLEAR$ = PREFIX$ + '2J'             !and to clear the screen
130 !
132 ! Escape sequences to move cursor and erase screen.
133 !
135 PRINT CLEAR$;                       !Clear screen
140 PRINT PREFIX$ + '16;4H';            !Move cursor to 16,4
160 PRINT 'Move the cursor to line 16, column 4 and print this text.'
170 SLEEP 3%
180 PRINT PREFIX$ + '1K';               !Erase text
186 PRINT PREFIX$ + '16;4H';            !Move cursor back to 16,4
200 PRINT PREFIX$ + '5A';               !Move cursor up 5 lines
220 PRINT 'Then move the cursor up 5 lines';
225 SLEEP 3%
230 PRINT CLEAR$;                       !Clear screen
250 PRINT PREFIX$ + '10C';              !Move cursor forward 10 spaces
270 PRINT 'and forward 10 spaces';
280 SLEEP 3%
290 PRINT CLEAR$;                       !Clear screen
300 PRINT PREFIX$ + 'H';                !Back to home position
310 !
320 ! Escape sequences for line size control and reverse video
350 !
370 PRINT PREFIX$ + '7m';               !Turn on reverse video
390 PRINT PREFIX$ + '16H' + ESC$ + '#3';
395 !Change line 16 to double-height top half
400 PRINT 'Double height line in reverse video';
410 !Change line 17 to double-height bottom half
420 PRINT PREFIX$ + '17H' + ESC$ + '#4';
430 PRINT 'Double height line in reverse video';
450 SLEEP 3%
460 PRINT PREFIX$ + 'm';                !Turn off reverse video
470 PRINT CLEAR$                        !Clear screen
32767 END
```

Output Escape Sequences

When you send an escape sequence to a terminal, use the value CHR$(155%) for the escape character if the terminal is in normal output mode. Do not use CHR$(27%), which is the ASCII decimal code for the ESC character, unless you are using transparent control character mode. The system translates CHR$(27%) to CHR$(36%), the dollar sign ($) character. CHR$(155%), an ESC with the high order bit set, is a special value that prevents the system from translating the ESC character to a $ character. When you use CHR$(155%), it causes the real CHR$(27%) to be sent, allowing the terminal to interpret the transmitted escape sequence. See the section, "Transparent Control Character Output: RECORD 16384% and MODE 16384%" for more information.

In processing output escape sequences, the system counts the escape characters along with the other characters to be output. This causes lines to wrap prematurely on video terminals. To avoid this line wrap, open the terminal in MODE 4% (suppress automatic CR/LF).

Input Escape Sequences

Under RSTS/E, terminals can operate in either escape sequence mode or no escape sequence mode. DIGITAL recommends that you set escape sequence mode by using MODE 256% in the OPEN statement (see the section "Escape Sequence Mode: MODE 256%"). For compatibility with existing applications, you can set either mode with the Set Terminal Characteristics SYS call (SYS 16), or the SET TERMINAL command (see the *RSTS/E System User's Guide*). New applications should use MODE 256%.

When a terminal is in normal mode, the system recognizes an incoming ESC character, CHR$(27%), as a delimiter and echoes a CHR$(36%), the $ character. When a terminal is in escape sequence mode, however, the system does special processing of input escape sequences. This special processing is useful for applications such as reading input from keypad function keys.

Note

> To cause a terminal to send escape sequences instead of numbers when keypad keys are pressed, you must send an escape sequence to the terminal. For the VT100 in ANSI-compatible mode, this escape sequence is "ESC=". See the appropriate hardware manual for details.

When a terminal is in escape sequence mode, the system processes input escape sequences so that:

- The characters in the escape sequence do not echo on the terminal

- A BASIC-PLUS program can read and test escape sequences

Input escape sequences are processed after CTRL/S and CTRL/Q (if the TTSYNC characteristic is set) but before private delimiters and all other characters. In brief, the system moves the ESC character from the beginning to the end of the escape sequence so that BASIC-PLUS can recognize the ESC character as a delimiter. The program receives the escape sequence as follows:

1. A CHR$(128%) value
2. The characters in the ESC sequence (minus the ESC character that started the sequence) without normal data conversions
3. A CHR$(155%) value, which signals the end of the escape sequence

Figure 4-1 shows an example of this conversion process.

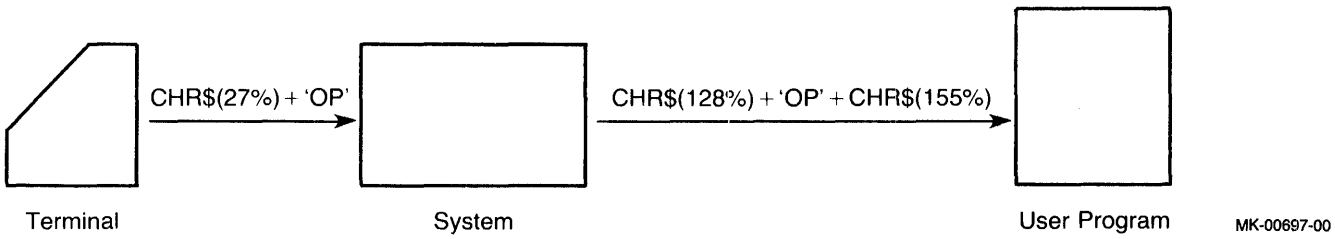


Figure 4-1: Input Escape Sequence Processing

Use GET statements to read incoming escape sequences, not INPUT or INPUT LINE statements. Unlike INPUT and INPUT LINE, GET does not strip the high order bit or discard nulls.

It is also a good idea to cancel type ahead right after you change a terminal's escape sequence characteristic or open a terminal in escape sequence mode (see SYS call 11, Cancel All Type Ahead). Canceling type ahead makes sure that the terminal's type ahead buffer does not contain a mixture of data processed in normal and escape sequence modes.

VT52 and VT100 ANSI-compatible escape sequences are defined so that matching keys on each terminal send escape sequences that end with the same character. Thus, you can use the same code to read incoming escape sequences from both terminals, regardless of whether the VT100s are in ANSI- or VT52-compatible mode.

For example, the up arrow key on a VT52 terminal (and a VT100 terminal in VT52-compatible mode) sends the sequence ESC+"A". Your program receives this sequence as CHR$(128%)+"A"+CHR$(155%). The up arrow key

on a VT100 terminal in ANSI mode sends the sequence ESC+"[A"; your program receives this sequence as CHR$(128%)+"[A"+CHR$(155%). By checking for an "A", your program can recognize the up arrow key from both terminals. Incoming escape sequences for other keys follow the same pattern. When you use this technique:

- Use MODE 256% to set escape sequence mode instead of setting the terminal's escape sequence characteristic. The system handles escape sequences that end with P differently for each method. See the section "Escape Sequence Mode: MODE 256%" for more information.

- Remember that it works only for reading incoming escape sequences; on output, your program must distinguish between a VT100 and a VT52. See the section "VT100- and VT200-Family Escape Sequences" for more information.

The rest of this section provides more detailed information on how the system processes escape sequences in escape sequence mode.

In escape sequence mode, an incoming ESC character CHR$(27%) sets a flag indicating that an escape sequence follows. The system does not echo the ESC character as a $ character and does not echo other characters in the sequence except for certain control characters. The terminal handles the characters in the escape sequence as follows:

1. The ASCII control characters (CHR$(0%) through CHR$(31%) and CHR$(127%)) are processed first. Except for DELETE (CHR$(127%)) and CTRL/U (CHR$(21%)), their functions do not change. The terminal discards DELETE and CTRL/U and does not passed them to the user. The control character CHR$(27%) (escape) starts a new escape sequence.

   Note that control characters in escape sequences violate the ANSI standard and should not be used.

2. Normal data conversion, such as translating lowercase letters to uppercase letters, is not done for characters inside an escape sequence.

3. The system resumes normal data conversions after it terminates the escape sequence.

Table 4-5 describes how the system terminates the escape sequence when it receives one of the escape sequence terminators.

Table 4-5: Escape Sequence Terminators

Sequence	Examples	Comments
Y<2 characters>	<ESC>Y<line#><col#>	The VT52 terminal uses this escape sequence for direct cursor addressing.
O<modifier> ?<modifier>	<ESC>OP <ESC>?M	The modifier can be any character except a control character. VT52 and VT100 terminals transmit escape sequences of this type when the terminal is in keypad application mode and a keypad key is pressed.
P<modifier>	--	The modifier can be any character except a control character. The system recognizes this sequence as an escape sequence terminator when you set the terminal's ESC SEQUENCE characteristic but not when you open the terminal with MODE 256%. See the section, Escape Sequence Mode: MODE 256%.*
P	<ESC>P <ESC>OP	The system recognizes P as an escape sequence terminator when you open the terminal with MODE 256% but not when you set the terminal's ESC SEQUENCE characteristic. See the section, Escape Sequence Mode: MODE 256%.*
[<n fillers><terminator>	<ESC>[5A <ESC>[10;15H	The filler characters must be in the range CHR$(32%) through CHR$(63%). The terminator character must be in the range CHR$(64%) through CHR$(128%). These are ANSI-compatible escape sequences.

Table 4-5: Escape Sequence Terminators (Cont.)

Sequence	Examples	Comments
<n fillers><terminator>	<ESC>#4 <ESC>= <ESC>Q	The filler characters must be in the range CHR$(32%) through CHR$(47%). The terminator character must be in the range CHR$(48%) through CHR$(126%). These are ANSI-compatible escape sequences. Some VT52 escape sequences, such as <ESC>Q (red key), are also recognized by this rule.

* As an alternative to MODE 256%, an optional patch is available that causes the system to recognize "P" as an escape sequence terminator. Unlike MODE 256%, this patch affects all terminals on the system. See the *RSTS/E Maintenance Notebook* for details.

The system starts another escape sequence whenever it receives another ESC character. If the ESC character precedes or is embedded in one of the character sequences in Table 4-5, the system does not append the CHR$(155%) value to the escape sequence it was processing before it starts processing the next one.

Transparent Control Character Output: RECORD 16384% and MODE 16384%

Until recently, most terminals had a character set of 128 characters. The characters were stored as 8 bits of data and were usually transmitted that way as well. The top bit (sign bit) of the 8-bit byte was always zero.

Now, many terminals support the international character set of 256 characters. For these terminals, all 8 data bits are significant. You can set the terminal to correctly handle the 256-character set using the /EIGHT_BIT qualifier of the SET TERMINAL command. See the *RSTS/E System Manager's Guide* for details.

RSTS/E terminal output processing normally modifies control characters in a variety of ways. For example, the terminal prints many characters with up-arrows, and converts ESC to $. To suppress these conversions, programs can add 128 to the value of the character to be printed. However, this is often inconvenient, especially in programs

that must also run on other operating systems. It also causes additional problems on 8-bit terminals.

On 8-bit terminals, the characters in the range 128-159 are called "C1 control characters" and have a different meaning from the corresponding characters with the sign bit cleared. Since RSTS/E normally assumes that characters in the range 128-159 are used to represent "real" control characters in the range 0-31, the new C1 control characters are not normally available.

Transparent control character output solves these problems. You specify it by using MODE 16384% in the OPEN statement, or by using the RECORD 16384% modifier in the PRINT or PUT statements. For example:

```
PUT #1%, RECORD 16384%
```

Transparent control character output is, in a sense, an intermediate form between "normal" and "binary" output. It processes the backspace, tab, line feed, vertical tab, form feed, and carriage return control characters in the usual way (for example, if the No Tab characteristic is set, tab expansion is performed). It transmits all other control characters unchanged, including the C1 control characters. Character codes 27 (ESC) and 155 (CSI) reset the position counter (CCPOS function value) to zero. Other control characters do not affect the position counter at all. Graphic (printable) characters are output in the same way as normal output.

Private Delimiters

A "private delimiter" is a character used as a delimiter within a program. You can define any printing or nonprinting character to be a private delimiter. For example:

- A letter
- A function key, such as DELETE
- A control character, such as CTRL/Z
- A standard delimiter, such as LINE FEED

A private delimiter is useful on a data entry terminal with a specialized keyboard. You can use a large or conveniently located key as the delimiter key. Private delimiters are also useful in keypad applications.

You can declare one character as a private delimiter on any RSTS/E system. Use the Set Terminal Characteristics SYS call (SYS 16), or the .SPEC directive (see the *RSTS/E System Directives Manual*).

Some RSTS/E systems allow the use of multiple private delimiters. If your system has this feature, you can declare up to 256 private delimiters with the .SPEC directive, available through MACRO. Multiple private delimiters let you do special character processing without using single character I/O. For example, by combining escape sequences with private delimiters, you can define your own function keys in keypad applications.

The .SPEC directive lets you set, read, and clear multiple private delimiters. You cannot set or read multiple private delimiters in BASIC-PLUS. For more information about the .SPEC directive, see the *RSTS/E System Directives Manual*. The rest of this section provides general information about private delimiters for both BASIC-PLUS and MACRO programmers.

Characteristics of Private Delimiters

When you declare a character as a private delimiter with either the Set Terminal Characteristics SYS call (SYS 16) or the .SPEC directive, it overrides the existing ASCII code for the character. Thus, unlike a standard delimiter such as RETURN or LINE FEED, a private delimiter does not echo at the terminal. In addition, a special character no longer performs its normal function. For example, when the DELETE key is a private delimiter, it does not erase the last character typed.

A private delimiter has basically the same characteristics as a standard delimiter. Like a standard delimiter, it:

- Terminates a read operation.
- Cannot be deleted (except with CTRL/X). The DELETE key and CTRL/U do not affect private delimiters in the type ahead buffer.
- Causes the system to awaken a sleeping job when typed at a terminal that the job has open or assigned. If the job cannot be awakened, the system stores the private delimiter character.

Once set, a private delimiter remains in effect for a terminal until either:

- The program clears it.
- The job releases the terminal by deassigning it or by closing the I/O channel where the terminal is open.

In addition, the system clears private delimiters when a dial-up line is hung up or the job controlling the terminal is killed.

Private delimiters change the way characters are processed in binary mode (MODE 1%). When a terminal is open in binary mode and no private delimiter is in use, the system terminates a read after every character. However, if one or more private delimiters are in use, the system terminates a read only when a private delimiter is typed.

The system processes private delimiters after processing CTRL/S and CTRL/Q (if the TTSYNC characteristic is set) and escape sequences (if the terminal is in escape sequence mode). This feature prevents a terminal from becoming permanently stalled, and it also lets you use private delimiters and escape sequences in the same program.

The system processes private delimiters before all other characters, including control characters (for example, CTRL/C). Thus, when you use a standard delimiter character as a private delimiter, it does not echo on the terminal.

Usage Notes for Private Delimiters

Follow these guidelines when using private delimiters:

- In a BASIC-PLUS program that uses a private delimiter, you must read input from the terminal with GET statements. Private delimiters do not work with INPUT, INPUT LINE, or MAT INPUT statements.

- By combining escape sequences with private delimiters, you can define your own function keys without using single character I/O. Follow these steps:

  1. Make sure the keypad is in the right mode for your application.

  2. Define each function as the PF1 key followed by a character.

  3. Define each character as a private delimiter so it does not echo on the terminal.

  For example, you might define PF1 + A as one function and PF1 + M as another function.

- To return a private delimiter character to its normal function, execute the Set Terminal Characteristics SYS call (SYS 16) or the .SPEC directive again. Note that while you can set and read multiple private delimiters only with the .SPEC directive, you can clear multiple private delimiters with either the .SPEC directive or the BASIC-PLUS SPEC% function (see the section "Private Delimiters").

Terminal Special Function: SPEC%

The SPEC% function performs special operations on terminals, pseudo keyboards (see Chapter 4), disks (see Chapter 1), flexible diskettes (see Chapter 1), magnetic tapes (see Chapter 2), and line printers (see Chapter 3).

For terminals, the SPEC% function allows you to cancel CTRL/O, set modes for tape, echo, and ODT, cancel type ahead, and clear private delimiters. The SPEC% function for terminals has the format:

```
VALUE%=SPEC%(FUNCTION%,PARAMETER,CHANNEL%,2%)
```

where:

VALUE%	depends on the 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: 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. FUNCTION%=9 Clear all private delimiters.
PARAMETER	specifies the terminal on which the operation is to take place. If PARAMETER is 0, the system performs the operation on the currently open terminal. If you specify a keyboard number in PARAMETER, the system performs the operation on that terminal. Note that you must allocate the keyboard to the calling job but you must not open it.
CHANNEL%	specifies the I/O channel for the terminal in PARAMETER.
2%	is the handler index for terminals.

Pseudo Keyboards

A pseudo keyboard is a logical device that has the characteristics of a terminal but has no terminal associated with it. Like a terminal, a pseudo keyboard has an input buffer and an output buffer, both of which come from the small buffer pool. User programs can send input to and get output from these buffers.

Using a pseudo keyboard lets one job control other jobs on the system. Pseudo keyboards are especially useful for batch operations because they let you do terminal I/O without tying up a terminal.

The system manager sets the number of pseudo keyboards on the system during system installation. The system assigns a device name of PKn: to each pseudo keyboard and associates each one with a keyboard unit number KBn: but not with a physical terminal. For example, the system may associate PK5: with KB8: even though no physical keyboard 8 exists.

Using a pseudo keyboard involves a controlling job and a controlled job. The controlling job (your program) creates the controlled job and then does I/O to it through the pseudo keyboard, PKn:. You can run LOGIN and use both system 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 KBm: associated with PKn:). However, the controlled job does not know it is working with a pseudo keyboard. Instead, it does input and output on its own keyboard, KB:.

Figure 4-2 shows the interaction between the controlled and controlling jobs.

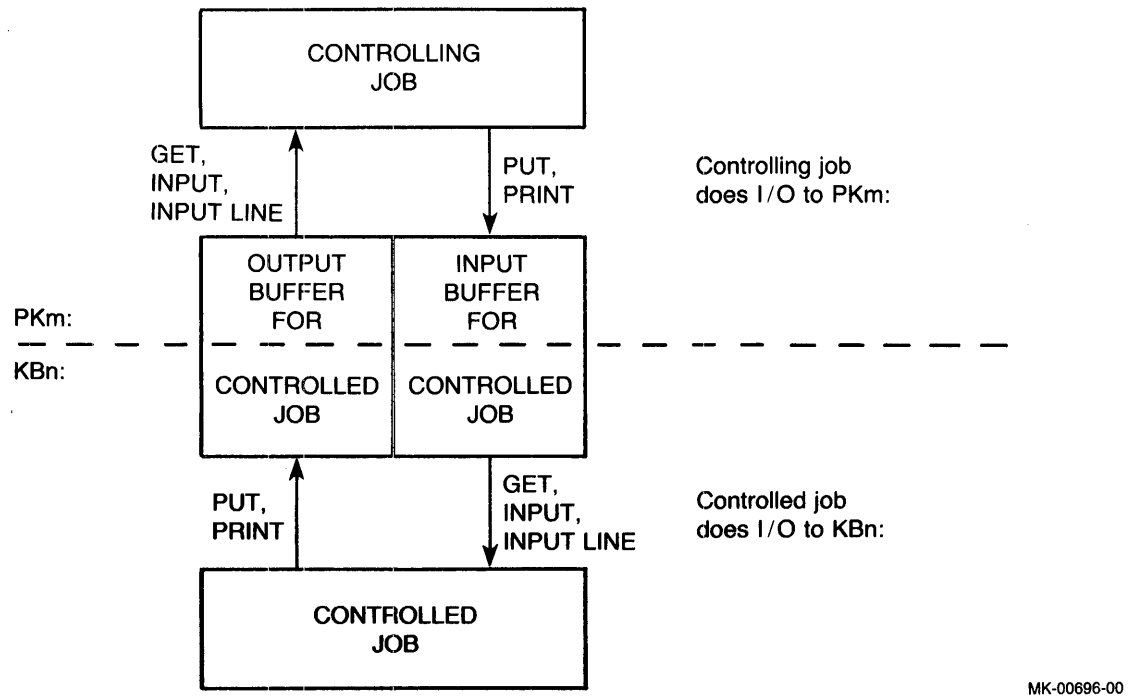


Figure 4-2: Pseudo Keyboard Operations

The system transfers data to a pseudo keyboard in full duplex mode. This means that strings sent by PUT or PRINT statements are echoed in the output buffer of the associated keyboard unit. Your program can read this echo with GET, INPUT, or INPUT LINE statements. In addition, when you send a carriage return character (CHR$(13%)) to the controlled job's input buffer, the system automatically appends a line feed character.

The rest of this section contains the following pseudo keyboard information:

- How to access a pseudo keyboard, create a controlled job, and perform pseudo keyboard I/O
- A sample program
- The SPEC% function for pseudo keyboards

Accessing the Pseudo Keyboard

Use the OPEN statement to access a pseudo keyboard. For example:

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

This OPEN statement associates pseudo keyboard unit 0 with I/O channel 1 and sets up its input and output buffers. Use this simple form of the OPEN statement; the system ignores the optional phrases FOR INPUT and FOR OUTPUT when opening pseudo keyboards.

Two MODE values are available for pseudo keyboards. MODE 0%, the default, causes the system to kill the controlled job when you close the pseudo keyboard. MODE 1% requires EXQTA privilege and causes the system to detach the controlled job when you close the pseudo keyboard. 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 system kills the job running on PK3: and detaches the job running on PK5:.

When the PK side of a pseudo keyboard is open, its KB side functions like a real keyboard. It can be opened, closed, assigned, and deassigned. You can broadcast data to it and force input to it. However, when the PK side of a pseudo keyboard is not open, its KB side functions like a disabled terminal. The system does not process input from it or send output to it. See the Disable Terminal SYS call (SYS 8) for more information about disabled terminals.

Two errors can occur when you open a pseudo keyboard:

- If the device you specify does not exist on the system, the error ?Not a valid device (ERR=6) occurs.
- If another job has the device assigned or opened, the error ?Device not available (ERR=8) occurs.

Creating the Controlled Job

After you open a pseudo keyboard, you must start the controlled job. The normal way to create the controlled job is with the Create A Job SYS call, (SYS 24). In some cases, you could force the LOGIN dialogue instead, but that requires you know the account password.

After the controlled job is running, you can send system commands, program commands, and program responses to the PK device by using PUT or PRINT statements with various RECORD options. Use GET statements to obtain output from the controlled job. The next section explains pseudo keyboard I/O in detail.

Pseudo Keyboard I/O

Reading from a pseudo keyboard is the same as reading from the controlled job's screen; writing to a pseudo keyboard is the same as typing at the controlled job's terminal or forcing input to the controlled job's keyboard.

Pseudo Keyboard Input

To obtain output from the controlled job, execute a GET statement on the I/O channel where the pseudo keyboard is open. For example, the following statement transfers data from the controlled job's output buffer to your program's channel 1 buffer:

```
100 GET #1%
```

The system never stalls the controlling program to wait for data. Instead, it immediately returns the contents of the controlled job's output buffer to the controlling job. The buffer contents may be a single message, several messages, or a message fragment. If no input is available, the error ?End of file on device (ERR=11) occurs.

If the controlled job performs output faster than the controlling job can execute GET statements, the keyboard output buffer fills. As a result, 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 controlled job's output.

Pseudo Keyboard Output

To perform output to a pseudo keyboard, execute a PRINT or PUT statement with a coded value in the RECORD option. For example:

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

where:

N% is the I/O channel where the PK device is open

C% is the number of bytes to send from the I/O buffer to the controlled job's input buffer.

If you omit the COUNT option, the PUT statement sends either 128 bytes (the pseudo keyboard's default buffer size) or the number of bytes specified in the RECORDSIZE option of the OPEN statement.

R% determines the actions the system performs for a specific PRINT or PUT statement. R% is an integer whose value the system interprets on a bit-by-bit basis. The system tests the low order four bits in R% (the bits numbered 0 through 3 from right to left) and executes the PRINT or PUT statement depending on whether certain bits are on or off.

Figure 4-3 explains the bit tests.

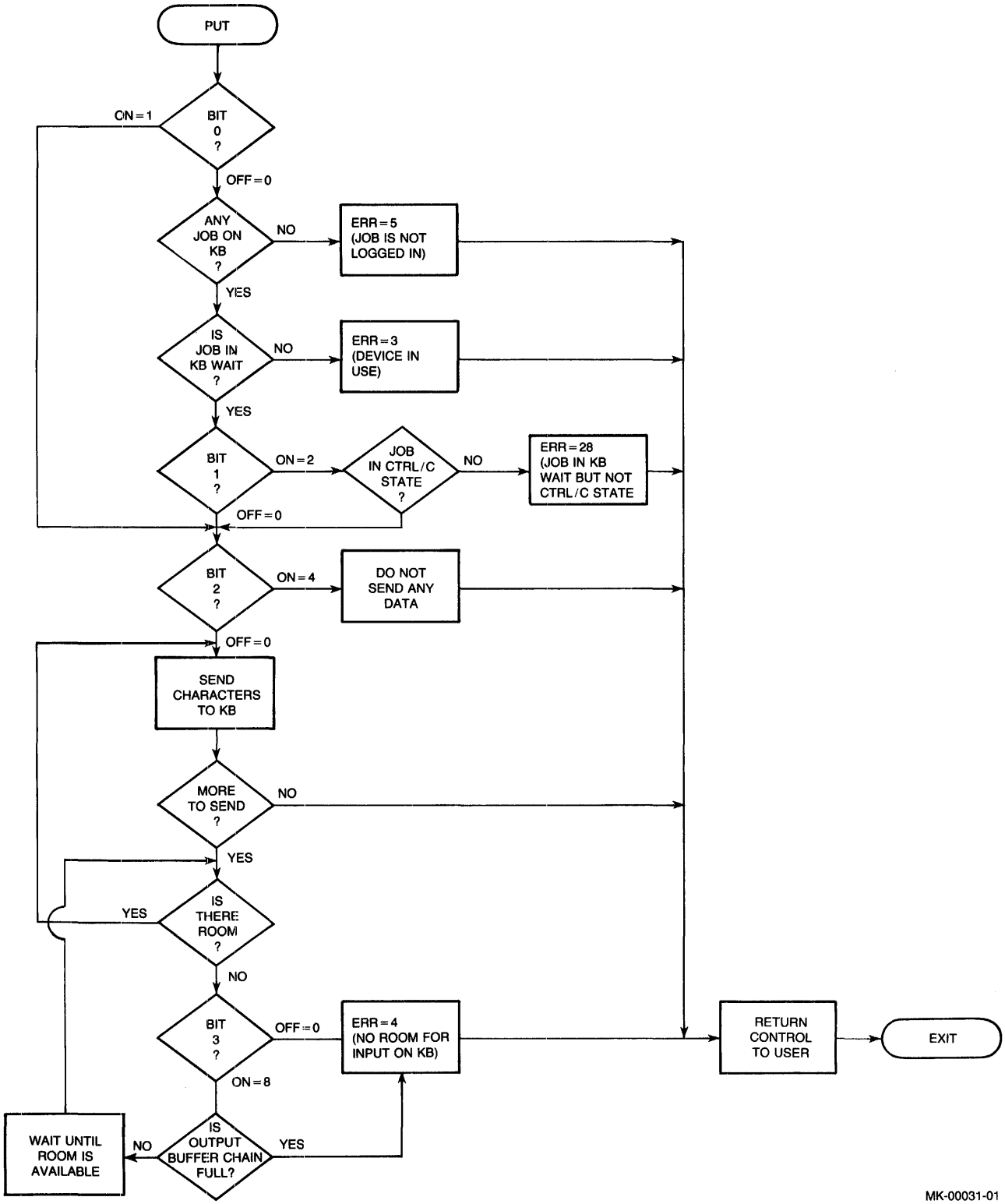


Figure 4-3: PUT Statement Actions for Pseudo Keyboard Output

Figure 4-3 shows the actions the system performs by testing the bits in R%. In summary:

Bit 0 (value = 1)	If set, the system does not check job status before sending data to the pseudo keyboard.
Bit 1 (value = 2)	If set, the system tests whether the pseudo keyboard is waiting for a system command (^C state) or is waiting for program input (KB wait state).
Bit 2 (value = 4)	If set, the system does not send data to the pseudo keyboard but instead returns control to the controlling program.
Bit 3 (value = 8)	If set, and there are no small buffers for keyboard input, the system waits until small buffers are available. However, your program receives an error if the output buffer chain is full.

The data you send to a pseudo keyboard must have the same format as data typed at a keyboard. For example, if you send a line that would normally end with the RETURN key, you must end the line with a carriage return character (CHR$(13%)). In addition, the value you specify in the COUNT option must include the carriage return character. Do not end the line with a carriage return/line feed sequence; the system automatically appends a line feed character to a line that ends with a carriage return character (just as it does when you enter a line at a terminal with the RETURN key).

Your program should send only one line at a time and retrieve each program or system response separately. Sending multiple lines fills up small buffers. For the same reason, the user should not type ahead. In addition, do not send a line unless the PK device is waiting for input. Always check PK device status before sending data.

Use the RECORD 6% option (values 2 and 4) in a PUT or PRINT statement to ensure that the controlled job is at command level. If the job is waiting for keyboard input but is not at command level, the error ?Programmable ^C trap (ERR=28) occurs. You must force a CTRL/C to the controlled job; otherwise, control returns to your program, which can then send a system command.

To run a program under the controlled job:

1. Use a PUT or PRINT statement with the RECORD 6% option to make sure that the controlled job is at command level.
2. Send the RUN command followed by the program name to the PK device.

The RECORD 16% option lets you kill any job currently running on the pseudo keyboard. In the PUT statement, specify the I/O channel where the pseudo keyboard is open. For example:

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

This statement kills the job currently running on the PK: unit open on channel 8.

Pseudo Keyboard Escape Sequence Processing

When you output escape sequences on a pseudo keyboard, the terminal driver translates CHR$(155%) to an escape ESC character (ASCII 27). The translation is necessary to properly handle eight bit terminal input. ATPK takes that pseudo keyboard output and displays it on your terminal. However the terminal driver now translates the ESC character to a $ character.

To make pseudo keyboard processing work correctly when using escape sequences, use either binary MODE (1%), or transparent control character output MODE (16384%) as an open mode, or use the RECORD 4096% modifier. This allows the terminal driver to correctly read escape characters back from the pseudo keyboard (without translation). See the sections "Binary Data Output and Input: RECORD 4096% and MODE 1%" and "Transparent Control Character Output: RECORD 16384% and MODE 16384%" for more information.

Programming Example

The following sample program uses a pseudo keyboard to process a command file:

```
 10  EXTEND
100 OPEN "PK8:" AS FILE #1%
110 PRINT "What command file do you want to use";
120 INPUT LINE FILENAME$
130 OPEN FILENAME$ FOR INPUT AS FILE #2%
140 PRINT "What is the account to log into";
150 INPUT LINE PPN$
160 PPN$ = CVT$$(PPN$,4%)
170 INPUT "What is the password"; PW$
180 PRINT #1%, RECORD 1%, "HELLO "; PPN$; CHR$(13%);
190 PRINT #1%, RECORD 1%, PW$ + CHR$(13%);
200 ON ERROR GOTO 19000
210 SLEEP 1%
220 GET #1%
230 FIELD #1%, RECOUNT AS A$
240 PRINT A$;
```

```
250 GOTO 220
260 PRINT #1%, RECORD 4%
270 INPUT LINE #2%, B$
280 B$ = CVT$$(B$,4%)
290 PRINT #1%, B$; CHR$(13%);
300 GOTO 220
19000 IF ERR = 11% AND ERL = 220% THEN RESUME 260 &
      ELSE IF ERR = 3% AND ERL = 260% THEN RESUME 210 &
      ELSE IF ERR = 11% AND ERL = 270% THEN RESUME 19100 &
      ELSE ON ERROR GOTO 0
19100 CLOSE #1%, #2%
32767 END
```

Line 100 opens the pseudo keyboard on I/O channel 1. Lines 110 through 170 ask the user (the controlling job) for a command file name and accounting information for the controlled job. Lines 180 and 190 create the controlled job by sending LOGIN input to the pseudo keyboard. Both PRINT statements use RECORD 1% to tell the system not to check job status before sending data.

The next section of the program consists of two loops:

- The first loop (lines 220 through 250) repeatedly gets data from the controlled job's output buffer and prints it on the controlling job's terminal. When there is no more data in the buffer, control goes to the error handling routine at line 19000.

- The second loop (lines 260 through 300) first uses a PRINT statement with RECORD 4% to see if the controlled job is waiting for keyboard input. If it is, the program reads a line from the command file and sends it to the controlled job's input buffer. Control then goes back to the first loop.

  If the controlled job is not waiting for keyboard input, control goes to error handling routine.

The error handling routine (lines 19000 through 19100) processes two errors:

- ?End of file on device (ERR=11). This error can occur for two different reasons in this program:

  - The controlled job's output buffer is empty.

  - There are no more commands in the command file.

  If the output buffer is empty, control goes to the loop that reads the next command from the command file. If there are no more commands in the command file, the program closes I/O channels and ends.

- ?Account or device in use (ERR=3). This error occurs if the controlled job is busy (that is, not waiting for keyboard input) when the program checks to see if it is ready for another command. The error handling routine transfers control to the SLEEP statement at line 210, which suspends program execution for one second before starting to execute the first loop again. The program works without the SLEEP statement but makes less efficient use of system resources.

Pseudo Keyboard Special Function: SPEC%

The SPEC% function performs special operations on pseudo keyboards, terminals (see Chapter 4), disks (see Chapter 1), flexible diskettes (see Chapter 1), magnetic tapes (see Chapter 2), and line printers (see Chapter 3).

For pseudo keyboards, the SPEC% function lets you:

- Disable and enable echo at the controlled job's keyboard (that is, the KB side of the pseudo keyboard)
- Read a flag word that tells you whether echo is ON or OFF at the controlled job's keyboard
- Read the current exit status of the job you are controlling.

A pseudo keyboard receives two kinds of output from a controlled job: character echo, which is done by the RSTS/E monitor, and program output, which occurs when a program writes to the controlled job's keyboard. The SPEC% function affects only character echo, not program output.

Character echo is enabled by default. However, in some pseudo keyboard applications it is more convenient to disable character echo. For example, in a pseudo keyboard application that uses both a terminal and a pseudo keyboard, you get character echo from the terminal; you also get character echo and program output from the pseudo keyboard. You can use this function to disable character echo at the pseudo keyboard.

The SPEC% function for pseudo keyboards has the format:

```
VALUE% = SPEC%(FUNCTION%, PARAMETER%, CHANNEL%, 16%)
```

where:

VALUE% depends on the function code you specify in FUNCTION%.

FUNCTION%=0% a flag word that contains information about the controlled job's keyboard. By testing bit 5 in VALUE%, you can determine whether keyboard echo is enabled or disabled. The tests are:

VALUE% AND 32% <> 0%
Keyboard echo is disabled.

VALUE% AND 32% = 0%
Keyboard echo is enabled.

FUNCTION%=1% returns the current exit status and the worst exit status for the job you are controlling:

VALUE% AND 7%
The current exit status, from the list below.

(VALUE%/16%) AND 7%
The worst exit status the job has had, from the list below.

Value	Status
0%	Warning
1%	Success
2%	Error
4%	Severe error

PARAMETER% depends on the function code you specify in FUNCTION%.

FUNCTION%=0% specifies the operation to perform:

Value	Operation
0%	Read the flag word
255%	Enable echo
-1%	Disable echo

FUNCTION%=1% unused

CHANNEL% specifies the I/O channel where the pseudo keyboard is open.

16% is the device handler index for pseudo keyboards.

Chapter 5

DECtape, Paper Tape, and Card Reader

This chapter describes how to process TU56 DECtape as a file-structured and a non-file-structured device. This chapter also describes the use of paper tape and card readers on RSTS/E.

File-Structured DECtape: TU56

To indicate file-structured processing on TU56 DECtape, include a file name with the device specification in the OPEN statement. Up to 15 files can be open for read access simultaneously on a TU56 DECtape drive. However, only one file can be open for write access. An attempt to open a second file for write access while one is currently open generates the error ?Too many files open on unit (ERR=17).

When a file is opened on TU56 DECtape, RSTS/E implicitly allocates the unit to the job performing the OPEN operation. Another job attempting to access the DECtape receives the error ?Device not available (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 two bytes as a pointer to the next block in the file. (See Figure 5-1.) GET and PUT statements read and write successive blocks on the tape. You cannot use the RECORD option to access blocks in the file in a random manner.

If you specify the RECORDSIZE option in the OPEN statement, the system creates a buffer of the value given in the option. 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 error ?Illegal byte count for I/O (ERR=31).

Non-File-Structured DECtape: TU56

In non-file-structured processing of TU56 DECtapes, you can access specific physical blocks on the DECtape. To start non-file-structured processing, specify only a device designator in the OPEN statement. Only OPEN FOR INPUT and OPEN are valid. The following two statements are equivalent because both reading and writing of physical blocks on the device are permitted:

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

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

BASIC-PLUS creates a 512-byte buffer. The following statement is invalid because it attempts to create a file:

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

After opening a TU56 DECtape device for non-file-structured processing, you can use the RECORD option in GET and PUT statements to retrieve and write specific physical blocks on the device. The record number you specify 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, the following statement reads block 4 of the file opened on channel 1% backwards:

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

The maximum block number is 577.

In writing non-file-structured TU56 DECtape files, you can specify how blocks should be accessed. Whenever possible, the system writes a file on a file-structured DECtape on every fourth block (that is, block N, N+4, N+8, and so on) of the DECtape. 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 (see Figure 5-1) shows how non-file-structured DECtape access time can be minimized.

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

Figure 5-1 shows the TU56 DECtape format.

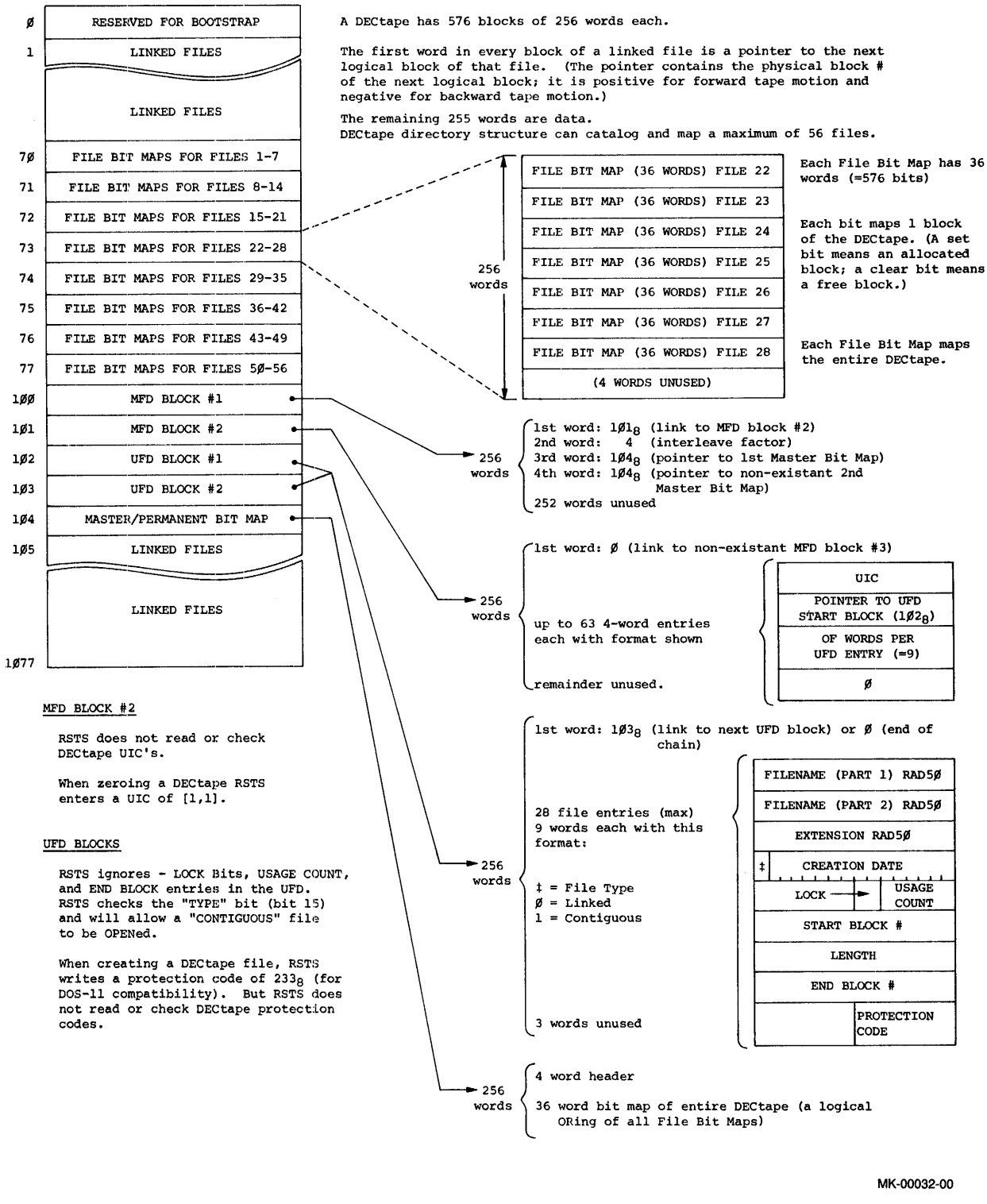


Figure 5-1: TU56 DECtape Format

Paper Tape

The paper tape reader and punch output data onto unoiled paper tape. In a punch operation, characters are read from the I/O buffer and translated into perforations on the tape. In a read operation, the tape perforations are translated back into characters and written to your I/O buffer. You can specify two modes to set even or odd parity for the characters punched on the tape and to check the parity of the characters read from the tape.

Punching with Parity on Paper Tape

Use the MODE option in an OPEN statement 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 eight-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.

Valid MODE values are:

```
MODE 0%  No parity punched
MODE 2%  Even parity
MODE 3%  Odd parity
```

Parity Checking on Paper Tape

You can also use the MODE option in an OPEN statement 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 eight-bit characters from the tape to the 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 seven-bit character to your buffer. If the parity does not match the specification, the driver writes a seven-bit character with the high-order bit set. After the tape is read, any characters in the buffer with bad parity return the error ?Data error on device (ERR=13).

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

Valid MODE values are:

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.

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, packed Hollerith, or binary. One card can be read (and the data on it stored) in any mode.

ASCII Mode: MODE 0%

The card reader reads cards punched with the standard ASCII codes, as shown in Appendix B. One of four sets of codes can be used: ANSI, 029, 026, or 1401. The code set for the system is specified during system installation. Cards punched in other formats are not acceptable to RSTS/E in ASCII mode. 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 the error ?End of file on device (ERR=11), 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 and 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$.

You can read cards with INPUT, INPUT LINE, or GET statements. If a card is misread or contains any illegal punches, the error ?Data error on device (ERR=13) occurs. With INPUT or INPUT LINE statements, any columns containing illegal punches are stored as BACKSLASH (ASCII 92) codes. If you read the card with a block 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 high order bit set). By checking the characters for code 220, your program can determine in which column(s) the error(s) occurred.

Packed Hollerith Mode: MODE 1%

In packed Hollerith read mode, the value of the RECOUNT variable is always 80, because 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.

Figure 5-2 shows the packed Hollerith read mode values

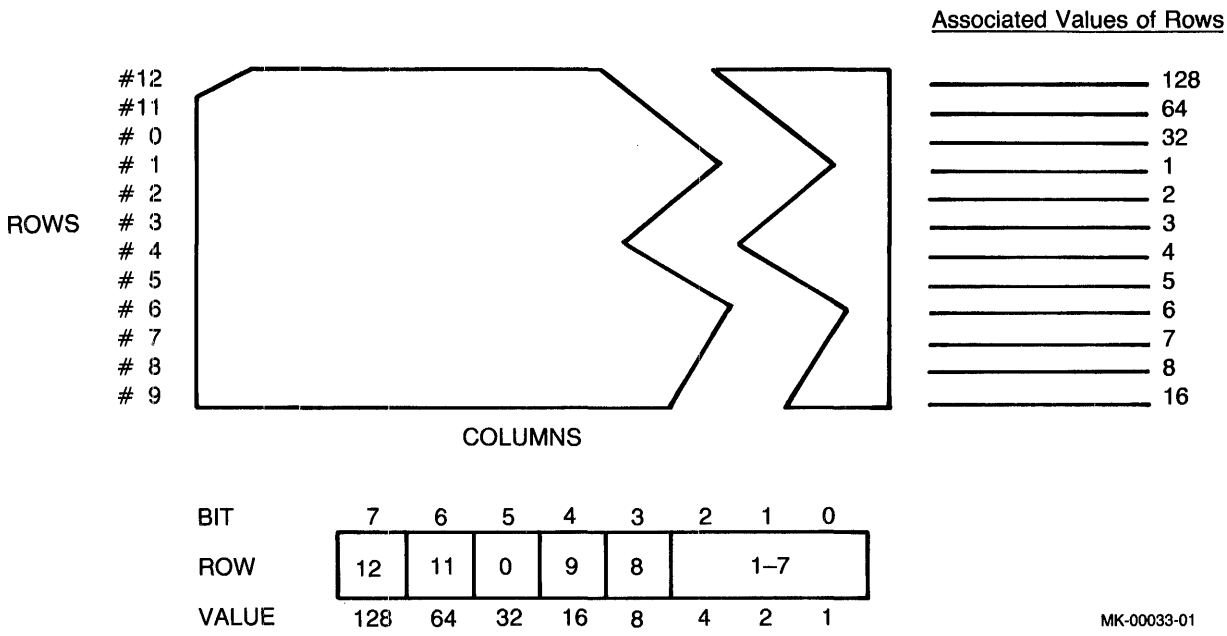


Figure 5-2: Packed Hollerith Read Mode

Note 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.

Binary Mode: MODE 2%

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.

Figure 5-3 shows the binary read modes.

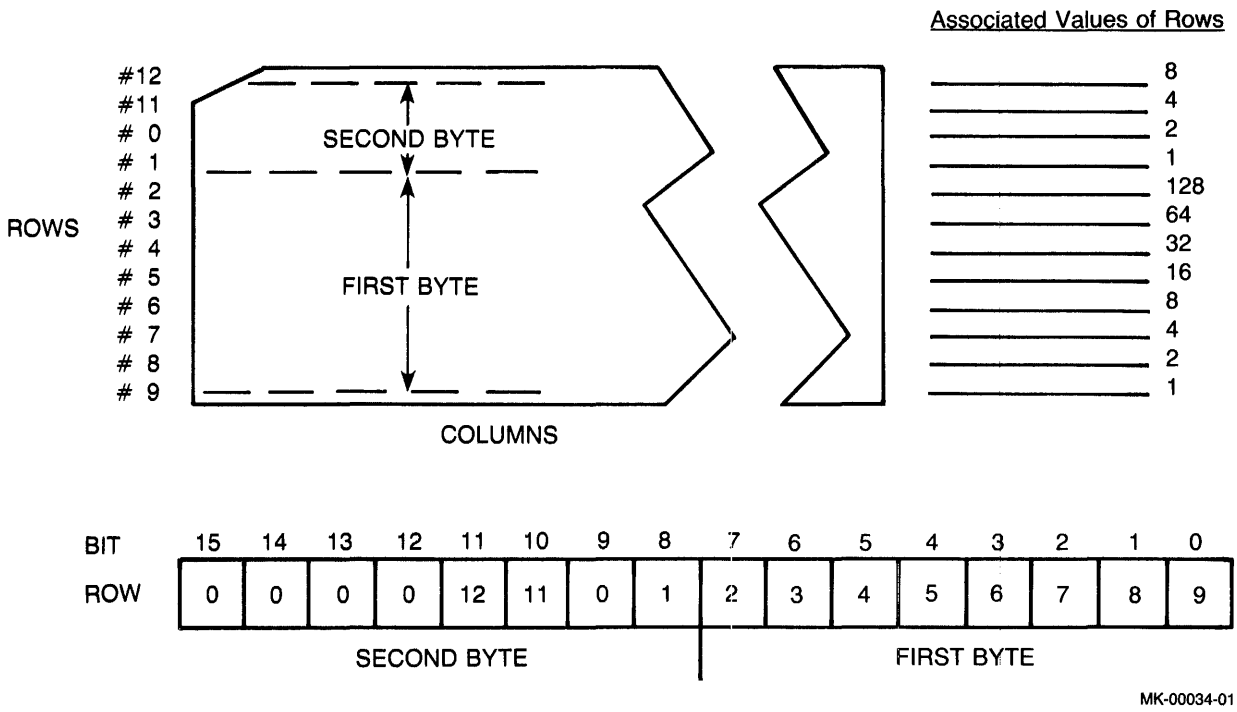


Figure 5-3: Binary Read Mode

Setting Read Modes

You can specify a read mode in an OPEN statement (with the MODE option) or a GET statement (with the RECORD option). Table 5-1 shows the MODE and RECORD values that correspond to each read mode. The default mode is 0 (ASCII).

As shown in Table 5-1, you must specify an explicit value when you use the MODE or RECORD option; failure to do so results in an error message.

Table 5-1: Specifying Read Modes on Card Reader

Statement	Option	Specified Read Mode
OPEN	MODE 0	ASCII
	MODE 1	Packed Hollerith
	MODE 2	Binary
GET	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 packed Hollerith read mode. Line 110 specifies binary read mode for 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. Consider the following sample program segment:

```
                                                    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%
  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 sample program sets the read mode to Hollerith and 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 reestablish a read mode.

Chapter 6

DMC11/DMR11 Interprocessor Link

This chapter describes how to use the DMC11 and DMR11 devices in a program to set up a communication link to another processor. Although the DMR11 differs in some details from the DMC11, they appear identical to your program.

Using the DMC11/DMR11 Interprocessor Link in Point-to-Point Configurations

The DMC11/DMR11 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/DMR11 is used by the DECnet/E package, which supports multiple node networks, user data security, multiple logical links over a single physical link, and other network features. When in use by DECnet/E, the DMC11/DMR11 is not available to you except through DECnet/E. However, in point-to-point configurations, DECnet/E may not be needed. In these cases, you can access the XM: device directly from a program to obtain a communication link with another processor. DECnet/E need not even be configured into the RSTS/E system.

The OPEN Statement

The DMC11/DMR11 is not a file-structured device. However, you must specify certain parameters at open time to establish the device's operating mode. In addition, when you execute an OPEN statement on a DMC/DMR with an autoanswer/autodial phone connection, the Data Terminal Ready (DTR) modem control signal is automatically raised to enable data transmission.

MODE Value

The MODE value used when opening a DMC11/DMR11 indicates whether the unit is to be run in full-duplex or half-duplex mode. These modes are described in the *Terminals and Communications Handbook*. To cause the DMC11/DMR11 to hang up a phone connection when it receives a DDCMP restart, add 512 to the specified MODE value. To specify full duplex, omit the MODE option in the OPEN statement, or specify a MODE value of zero. To specify half duplex, use a MODE value of 1024.

CLUSTERSIZE Value

To ensure that messages from the remote processor to the local RSTS/E system are received without need for retransmission, the DMC11/DMR11 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, called the receive complete queue. When you issue a GET statement on the open channel of the DMC11/DMR11, the message is copied from the system buffer to your I/O buffer, and the system buffer is released.

Because a buffer on the receive complete queue is no longer available for use by the DMC11/DMR11, 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 that the driver attempts to keep allocated to the DMC11/DMR11 receiver side is called the buffer quota for the unit. You specify it at open time as the CLUSTERSIZE value. Any number from 1 to 127 is valid as a buffer quota, but values above 4 are not recommended except when a very large volume of traffic on a high speed (higher than 56K baud) line is expected; allocating too many buffers to the DMC11/DMR11 needlessly ties up system resources. However, if the buffer quota is too low, overrun errors occur on the unit. These do not cause any loss of data, but do result in reduced performance due to retransmissions.

FILESIZE Value

The value used in the FILESIZE option at open time specifies the size of the buffers allocated to the DMC11/DMR11 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 is lost and the DMC11/DMR11 halts operation. Note that the 632-byte limit on receive buffer size does not limit the length of transmitted messages. The DMC/DMR driver limits transmitted message lengths to a maximum of 8000 bytes. However, to avoid message truncation, you 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/DMR11).

RECORDSIZE Value

The RECORDSIZE value establishes the I/O buffer size for the DMC11/DMR11. The default buffer size is 512 bytes. While you can specify any even buffer size, it is good practice to make the I/O buffer the same size or larger than the device's receive buffer (see the section, "The GET Statement and RECORD Options").

Errors

Only two errors specific to the DMC11/DMR11 can occur at open time. The error ?Device hung or write locked (ERR=14) occurs if the driver cannot initialize the device. The error ?No buffer space available (ERR=32) occurs if the driver cannot obtain a 264-byte buffer to use as the hardware base table.

The GET Statement and RECORD Options

The GET statement copies the next message from the DMC11/DMR11 queue of received messages into your program's I/O buffer. If the received message is longer than your buffer, the monitor truncates it with no warning. Therefore, it is good practice to specify a RECORDSIZE in the OPEN statement that is greater than or equal to the FILESIZE value (see the section, "FILESIZE Value").

The value in the RECORD option of GET statements determines how the program treats message unavailability. If no message is available and the DMC11/DMR11 is still running (that is, the physical link is intact), you can cause your job 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 the error ?Can't find file or account (ERR=5) immediately. A RECORD value of 8192% tells the monitor to stall the job until either:

- A message is available from the remote processor, in which case the message is returned in the user's buffer as usual
- A DMC11/DMR11 error occurs, in which case the program receives the error ?Device hung or write locked (ERR=14)

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 is awakened by any of the following conditions:

- A message is received on the DMC11/DMR11.
- An error occurs on the DMC11/DMR11.
- A message is received through the local send/receive mechanism.
- A delimiter is typed on one of the job's keyboards.
- The number of logins is set to 1.
- 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 the error ?Device hung or write locked (ERR=14). If it is awakened for any other reason, it receives the error ?Can't find file or account (ERR=5).

When the DMC11/DMR11 driver detects a failure in the physical link, it shuts down the unit. Any messages received before 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 the error ?Device hung or write locked (ERR=14). The only recourse is to close the channel on which the unit is open.

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/DMR11. Instead, it returns count and status information to the user. Twenty-six bytes of information are returned in the following format:

Bytes	Meaning
1	Number of transmit buffers actually being processed by the DMC11/DMR11 hardware.
2	Total number of transmit buffers waiting to be sent, including those given to the hardware (that is, number of uncompleted PUT statements).
3	Number of messages on the receive queue waiting to be given to the job.

4 Reserved.

5 Number of receive buffers actually given to the DMC11/DMR11 hardware.

6 Total number of buffers allocated to the DMC11/DMR11 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/DMR11 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-
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/message received.
7	Lost data error.
8	Reserved.
9	Disconnect code.
10	DDCMP start received.
11	UNIBUS address timeout on DMC/DMR access.
12	Procedure error.
255	Timeout error.

10 Status flags, encoded as a combination of bits:

4	The first transmit since the DMC11/DMR11 was opened is complete, indicating that a link has been established and that further transmits will 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/DMR11 was halted for the reason given in byte 9.

All other bits are reserved.

11-12 Receive buffer size (from the FILESIZE value when the DMC11/DMR11 was opened).

13-14 Operational mode (from the MODE value when the DMC11/DMR11 was opened).

15-16 Reserved.

17 Receive buffer quota (from the CLUSTERSIZE value when the DMC11/DMR11 was opened).

18 Reserved.

19-20 Value of SEL6 hardware register at most recent control-out interrupt. If the DMC11/DMR11 was halted due to a hardware error, the specific error (or errors) can be found here. For the format of this word, see the description of the DMC11 and DMR11 in the *DMR11 Synchronous Controller Users's Guide*.

21-22 Data check count. A data check error occurs when the DMC11/DMR11 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/DMR11 continues to retry the transmission and reports a data check error every seven retries. The total number of data check errors that have occurred since the DMC11/DMR11 unit was opened is returned in this word.

23-24 Timeout count. A timeout error occurs when the DMC11/DMR11 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 (the number of overrun errors since the OPEN). An overrun error indicates that a message was received but no buffer was available. This is nonfatal because the remote system retransmits the message (and possibly logs data check errors). You can reduce overrun errors by increasing the buffer quota for the unit (see CLUSTERSIZE in OPEN). Overrun errors can also occur when the driver is not able to obtain a buffer allowed by the buffer quota value. To reduce this type of overrun error, increase the size of XBUF at the start of the next time-sharing session.

Three errors (data check, timeout, and overrun) that are detected by the DMC11/DMR11 are only warnings. These are nonfatal and do not cause the unit to halt. Your program is not informed when they occur. However, if any of them occurs frequently, it indicates that the program has set the wrong CLUSTERSIZE value 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.

Your program never stalls when it issues a count and status request. Furthermore, this request is legal whether or not the unit is running. Thus, you can use it to determine the specific DMC11/DMR11 problem after the program receives an ERR 14.

The PUT Statement

The PUT statement copies data from your program'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 can be from 1 to 8000. A COUNT value outside that range generates the error ?Illegal byte count for I/O (ERR=31). If the monitor cannot obtain a buffer big enough to hold the message, it returns the error ?No buffer space available (ERR=32). 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 immediately returns the error ?Device hung or write locked (ERR=14). As with the GET statement, the only recourse is to close the channel.

The PUT statement queues messages to the DMC11/DMR11 to be sent as soon as possible. Your 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). You can modify this action by using the RECORD option of the PUT statement. A RECORD value of 0 (or omitting the RECORD option) tells the monitor to queue the data for transmission, 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/DMR11 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 is awakened by any of the following conditions:

- All pending transmissions have completed successfully.
- An error occurs on the DMC11/DMR11.
- A message is received through the local send/receive mechanism.
- A delimiter is typed on one of the job's keyboards.
- The number of logins is set to 1.
- N seconds have expired and n is not 0.

If the job is awakened for the second reason, it receives the error ?Device hung or write locked (ERR=14). If it is awakened for any other reason, it receives no error and continues processing normally. To find the number of transmissions still outstanding, use a GET with RECORD 4096% and examine the value in byte 2.

Adding the value 4096% to any of the above RECORD values tells the monitor not to transmit any data, but to do the WAIT operation specified.

The CLOSE Statement

If a DMC11/DMR11 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%+8192% 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/DMR with an autoanswer/autodial phone connection, the DTR (Data Terminal Ready) modem control signal is automatically dropped to disable data transmission.

Hardware Errors

Any fatal error detected by the DMC11/DMR11 (that is, any error not listed as nonfatal in the count and status description) causes the monitor to shut down the link. The monitor reports the error ?Device hung or write locked (ERR =14) to your 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.)

PART II
System Function Calls

Chapter 7

SYS System Function Calls

This chapter describes the system function calls, also known as SYS calls. System function calls let you perform many special functions, such as:

- Establish special characteristics for a job
- Perform special I/O functions
- Set terminal characteristics
- Modify account characteristics
- Manipulate account privilege information

The SYS call whose function code is 6 is a specialized case of the general system function call. SYS call 6 contains a subfunction code called the FIP code. The FIP code causes a dispatch call to be made to special resident or nonresident code that performs file processing. The subfunctions of SYS call 6 are called FIP calls. Because programmers generally use FIP calls more frequently than the SYS calls, the FIP calls are also commonly referred to as SYS calls. This chapter also uses SYS call as the preferred term.

The calls described in this Chapter are organized as follows:

- SYS system function calls (F=0 to F=14). The calls are arranged in ascending numerical order. Tables 7-1 and 7-2 summarize these calls.
- SYS system function calls to FIP (F0=-29 to F0=34). With two exceptions, the calls are arranged in ascending numerical order. Tables 7-3 and 7-4 summarize these calls.
- The PEEK function. This function lets a user who has RDMEM privilege examine any word location in the monitor part of memory.

SYS System Function Calls

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

The SYS call format is used for two reasons. First, 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 use of a variable number of parameters.

Some SYS calls provide one set of functions to a nonprivileged user, while providing the privileged user with a more powerful set. To find out what privileges are associated with each call, see Tables 7-1 through 7-4, as well as the individual description of each SYS call.

If you are not sure what privileges your account has, use the SHOW JOB/PRIVILEGES command to list them. If you have more privileges than you need to use a certain call and want to temporarily disable them, use the SET JOB/PRIVILEGES command. The DCL commands associated with privileges are described in the *RSTS/E System User's Guide*.

The first part of Chapter 7 describes all system function calls with function codes other than 6. The second part of Chapter 7 describes system function calls to the file processor (FIP calls). These calls are associated with system function call 6.

SYS System Function Formats and Codes

The general format of the SYS call is:

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

where:

V$ is the data (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.

F% in the general format denotes function codes that range from 0 through 14, inclusive. SYS calls that specify a code outside of this range or that pass a zero length string generate the error ?Illegal SYS() usage (ERR=18). Table 7-1, organized by code number, summarizes the codes and their functions. Table 7-2, organized alphabetically by function name, provides the same information.

The SYS call whose function code is 6 is a more specialized case of the general system function call. It is specialized by a subfunction code called the file processor (FIP) code. The FIP code causes a dispatch call to be made to special resident or nonresident 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 subfunction code.

O$ is the optional (by function code) parameter string passed by the call.

The section "SYS System Function Calls to FIP" describes the purpose, calling format, and use of each FIP system function call (F=6). It also describes how to build the parameter string to pass to the monitor and how to extract data from the returned string.

Table 7-3 in this section is a quick reference index of the FIP functions in order of FIP code (F0). Table 7-4 provides the same information, but is arranged alphabetically by function name. For detailed information on each of the functions, refer to the page shown beside the name in the table.

In Tables 7-1 through 7-4, the Relevant Privileges column lists the privileges associated with each SYS call. A user who attempts to call a SYS function without sufficient privilege receives the error ?Illegal SYS() usage (ERR=18) or the error ?Protection violation (ERR=10). To avoid repetition, this chapter describes error 18 for calls only if it has a meaning different from nonprivileged attempts to use the call.

Table 7-1: SYS System Function Calls (by Function Code)

Function Code(F)	Function Name	Relevant Privileges	Page
0	Cancel CTRL/O effect on terminal	None	7-18
1	Enter tape mode on terminal	None	7-19
2	Enable echoing on terminal	None	7-20
3	Disable echoing on terminal	None	7-21
4	Enable ODT submode on terminal	None	7-22
5	Exit with no prompt message	None	7-24
6	SYS call to the file processor	See individual FIP call	7-25
7	Get core common string	None	7-26
8	Put core common string	None	7-27
9	Exit and clear program	None	7-28
10	Reserved for special implementations	--	--
11	Cancel all type ahead	None	7-30
12	Return information on last opened file	None	7-31
13	Reserved for special implementations	--	--
14	Execute CCL command	Execute access to file	7-34

Chapter 7

SYS System Function Calls

This chapter describes the system function calls, also known as SYS calls. System function calls let you perform many special functions, such as:

- Establish special characteristics for a job
- Perform special I/O functions
- Set terminal characteristics
- Modify account characteristics
- Manipulate account privilege information

The SYS call whose function code is 6 is a specialized case of the general system function call. SYS call 6 contains a subfunction code called the FIP code. The FIP code causes a dispatch call to be made to special resident or nonresident code that performs file processing. The subfunctions of SYS call 6 are called FIP calls. Because programmers generally use FIP calls more frequently than the SYS calls, the FIP calls are also commonly referred to as SYS calls. This chapter also uses SYS call as the preferred term.

The calls described in this Chapter are organized as follows:

- SYS system function calls (F=0 to F=14). The calls are arranged in ascending numerical order. Tables 7-1 and 7-2 summarize these calls.
- SYS system function calls to FIP (F0=-29 to F0=34). With two exceptions, the calls are arranged in ascending numerical order. Tables 7-3 and 7-4 summarize these calls.
- The PEEK function. This function lets a user who has RDMEM privilege examine any word location in the monitor part of memory.

SYS System Function Calls

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

The SYS call format is used for two reasons. First, 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 use of a variable number of parameters.

Some SYS calls provide one set of functions to a nonprivileged user, while providing the privileged user with a more powerful set. To find out what privileges are associated with each call, see Tables 7-1 through 7-4, as well as the individual description of each SYS call.

If you are not sure what privileges your account has, use the SHOW JOB/PRIVILEGES command to list them. If you have more privileges than you need to use a certain call and want to temporarily disable them, use the SET JOB/PRIVILEGES command. The DCL commands associated with privileges are described in the *RSTS/E System User's Guide*.

The first part of Chapter 7 describes all system function calls with function codes other than 6. The second part of Chapter 7 describes system function calls to the file processor (FIP calls). These calls are associated with system function call 6.

SYS System Function Formats and Codes

The general format of the SYS call is:

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

where:

V$ is the data (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.

F% in the general format denotes function codes that range from 0 through 14, inclusive. SYS calls that specify a code outside of this range or that pass a zero length string generate the error ?Illegal SYS() usage (ERR=18). Table 7-1, organized by code number, summarizes the codes and their functions. Table 7-2, organized alphabetically by function name, provides the same information.

The SYS call whose function code is 6 is a more specialized case of the general system function call. It is specialized by a subfunction code called the file processor (FIP) code. The FIP code causes a dispatch call to be made to special resident or nonresident 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 subfunction code.

O$ is the optional (by function code) parameter string passed by the call.

The section "SYS System Function Calls to FIP" describes the purpose, calling format, and use of each FIP system function call (F=6). It also describes how to build the parameter string to pass to the monitor and how to extract data from the returned string.

Table 7-3 in this section is a quick reference index of the FIP functions in order of FIP code (F0). Table 7-4 provides the same information, but is arranged alphabetically by function name. For detailed information on each of the functions, refer to the page shown beside the name in the table.

In Tables 7-1 through 7-4, the Relevant Privileges column lists the privileges associated with each SYS call. A user who attempts to call a SYS function without sufficient privilege receives the error ?Illegal SYS() usage (ERR=18) or the error ?Protection violation (ERR=10). To avoid repetition, this chapter describes error 18 for calls only if it has a meaning different from nonprivileged attempts to use the call.

Table 7-1: SYS System Function Calls (by Function Code)

Function Code(F)	Function Name	Relevant Privileges	Page
0	Cancel CTRL/O effect on terminal	None	7-18
1	Enter tape mode on terminal	None	7-19
2	Enable echoing on terminal	None	7-20
3	Disable echoing on terminal	None	7-21
4	Enable ODT submode on terminal	None	7-22
5	Exit with no prompt message	None	7-24
6	SYS call to the file processor	See individual FIP call	7-25
7	Get core common string	None	7-26
8	Put core common string	None	7-27
9	Exit and clear program	None	7-28
10	Reserved for special implementations	--	--
11	Cancel all type ahead	None	7-30
12	Return information on last opened file	None	7-31
13	Reserved for special implementations	--	--
14	Execute CCL command	Execute access to file	7-34

Table 7-2: SYS System Function Calls (by Function Name)

Function Name	Function Code(F)	Relevant Privileges	Page
Cancel all type ahead	11	None	7-30
Cancel CTRL/O effect on terminal	0	None	7-18
Disable echoing on terminal	3	None	7-21
Enable echoing on terminal	2	None	7-20
Enable ODT submode on terminal	4	None	7-22
Enter tape mode on terminal	1	None	7-19
Execute CCL command	14	Execute access to file	7-34
Exit and clear program	9	None	7-28
Exit with no prompt message	5	None	7-24
Get core common string	7	None	7-26
Put core common string	8	None	7-27
Reserved for special implementations	10	--	--
Reserved for special implementations	13	--	--
Return information on last opened file	12	None	7-31
SYS call to the file processor	6	See individual FIP call	7-25

Table 7-3: FIP SYS Calls (by Subfunction Code)

Function Code(FO)	Function Name	Relevant Privileges	Page
-29	Get monitor tables - part III	None	7-58
-28	Spooling (Obsolete, use PBS request)	Read access to file Write access to file	7-61
-27	Snap shot dump	SYSIO	7-66
-26	File utility functions	Read access to file Write access to file DATES TUNE SYSIO	7-67
-25	Read/write file attributes	Read access to file Write access to file	7-75
-25	Read pack attributes	DEVICE	7-79
-25	Read/write account attributes	GACNT WACNT	7-81
-25	Delete account attributes	GACNT WACNT	7-87
-24	Add/delete CCL command	INSTAL	7-89
-23	Terminating file name string scan	None	7-45
-22	Set special run priority	TUNE	7-92
-21	Drop/regain (temporary) privileges	None	7-93
-20	Lock/unlock job in memory	TUNE	7-96
-19	Set number of logins	SWCTL	7-98
-18	Add run-time system	INSTAL	7-100
-18	Remove run-time system	INSTAL	7-104

Table 7-3: FIP SYS Calls (by Subfunction Code) (Cont.)

Function Code(FO)	Function Name	Relevant Privileges	Page
-18	Unload run-time system	INSTAL	7-106
-18	Add resident library	INSTAL	7-108
-18	Remove resident library	INSTAL	7-112
-18	Unload resident library	INSTAL	7-113
-18	Create dynamic region	INSTAL	7-114
-17	Name run-time system	Write access to file	7-117
-16	Shut down system	SHUTUP	7-119
-15	Accounting dump	GACNT WACNT	7-120
-14	Change system date/time	DATES	7-122
-13	Change priority/run burst/job size	TUNE	7-124
-12	Get monitor tables - part II	None	7-127
-11	Change file backup statistics	DATES	7-129
-10	File name string scan	None	7-45
-9	Hang up a dataset	HWCTL	7-132
-8	Get open channel statistics	None	7-134
-7	Enable CTRL/C trap	None	7-137
-6	Poke memory	SYSMOD	7-140
-5	Broadcast to terminal	SEND	7-141
-4	Force input to terminal	SYSIO	7-143
-3	Get monitor tables - part I	None	7-144
-2	Disable logins	SWCTL	7-147
-1	Enable logins	SWCTL	7-148

Table 7-3: FIP SYS Calls (by Subfunction Code) (Cont.)

Function Code(FO)	Function Name	Relevant Privileges	Page
0	Create user account (new format)	GACNT WACNT	7-149
0	Create user account (old format)	GACNT WACNT	7-154
1	Delete user account	GACNT WACNT	7-158
2	Reserved	--	--
3	Disk pack status	MOUNT HWCFG	7-160
4	Login	None	7-165
4	Verify password	DEVICE GACNT WACNT	7-165
5	Logout	EXQTA WACNT	7-169
6	Attach	GACNT WACNT	7-173
6	Reattach	DEVICE	7-175
6	Swap Console	None	7-177
7	Detach	JOBCTL	7-179
8	Change quota (old format)/expiration date/password (old format)	GACNT WACNT	7-182
8	Change quota (new format)/expiration date/password (old format)	GACNT WACNT	7-186
8	Set password (new format)	GACNT WACNT	7-188
8	Kill job	JOBCTL	7-190
8	Disable terminal	HWCTL	7-192

Table 7-3: FIP SYS Calls (by Subfunction Code) (Cont.)

Function Code(FO)	Function Name	Relevant Privileges	Page
9	Return error messages	None	7-194
10	Allocate/reallocate device	DEVICE HWCTL	7-196
10	Assign user logical	None	7-200
11	Deallocate a device or deassign user logical	None	7-202
12	Deallocate all devices	None	7-204
13	Zero a device	DEVICE Create/rename access to account	7-205
14	Read/read and reset accounting data	GACNT WACNT	7-209
15	Directory lookup on index	DEVICE Read or execute access to file	7-219
15	Special magnetic tape directory lookup	DEVICE	7-221
16	Set terminal characteristics - part I	HWCFG	7-229
16	Set terminal characteristics - part II	HWCFG	7-241
17	Disk directory lookup on file name	DEVICE Read or execute access to file	7-224
17	Disk wildcard directory lookup	DEVICE Read or execute access to file	7-226
18	Obsolete (use function code 22)	--	--
19	Enable/disable disk caching	TUNE	7-248

Table 7-3: FIP SYS Calls (by Subfunction Code) (Cont.)

Function Code(FO)	Function Name	Relevant Privileges	Page
20	Convert date and time	None	7-252
21	Add new logical names	INSTAL	7-256
21	Remove logical names	INSTAL	7-259
21	Change disk logical name	INSTAL	7-261
21	List logical names	None	7-263
22	Message send/receive	JOBCTL SEND SWCFG SWCTL SYSIO	Ch. 8
22	Send local data message with privileges	SEND	Ch. 8
22	Send Print/Batch Services request	None	Ch. 9
23	Add system files	Write access to file INSTAL	7-267
23	Remove system files	Write access to file INSTAL	7-270
23	List system files	None	7-272
24	Create a job	Execute access to file EXQTA JOBCTL TUNE WACNT	7-274
25	Wildcard PPN lookup	DEVICE	7-282
26	Return job status	JOBCTL TUNE	7-284
27	Reserved	--	--
28	Set/clear current privileges	None	7-289

Table 7-3: FIP SYS Calls (by Subfunction Code) (Cont.)

Function Code(FO)	Function Name	Relevant Privileges	Page
28	Read current privileges	None	7-289
29	Stall/Unstall system	HWCTL	7-291
30	Reserved	--	--
31	Third party privilege check	None	7-293
32	Check file access rights	None	7-295
32	Convert privilege name to mask	None	7-297
32	Convert privilege mask to name	None	7-299
33	Open next disk file	DEVICE Read access to file Write access to file DATES	7-301
34	Set device characteristics	HWCFG HWCTL	7-304
34	Set line printer characteristics	HWCFG	7-307
34	Set system defaults	HWCFG SWCFG	7-310
34	Load monitor overlay code and return status/remove monitor overlay code	SWCFG	7-313
--	PEEK function	RDMEM SYSMOD	7-317

Table 7-4: FIP SYS Calls (by Function Name)

Function Name	Function Code(FO)	Relevant Privileges	Page
Accounting dump	-15	GACNT WACNT	7-120
Add/delete CCL command	-24	INSTAL	7-89
Add new logical names	21	INSTAL	7-256
Add system files	23	Write access to file INSTAL	7-267
Add resident library	-18	INSTAL	7-108
Add run-time system	-18	INSTAL	7-100
Allocate/reallocate device	10	DEVICE HWCTL	7-196
Assign user logical	10	None	7-200
Attach	6	GACNT WACNT	7-173
Broadcast to terminal	-5	SEND	7-141
Change disk logical name	21	INSTAL	7-261
Change file backup statistics	-11	DATES	7-129
Change quota/expiration date/password	8	GACNT WACNT	7-182
Change priority/run burst/job size	-13	TUNE	7-124
Change system date/time	-14	DATES	7-122
Check file access rights	32	None	7-295
Convert date and time	20	None	7-252
Convert privilege mask to name	32	None	7-299
Convert privilege name to mask	32	None	7-297

Table 7-4: FIP SYS Calls (by Function Name) (Cont.)

Function Name	Function Code(FO)	Relevant Privileges	Page
Create a job	24	Execute access to file EXQTA JOBCTL TUNE WACNT	7-274
Create dynamic region	-18	INSTAL	7-114
Create user account (new format)	0	GACNT WACNT	7-149
Create user account (old format)	0	GACNT WACNT	7-154
Deallocate all devices	12	None	7-204
Deallocate a device or deassign user logical	11	None	7-202
Delete account attributes	-25	GACNT WACNT	7-87
Delete user account	1	GACNT WACNT	7-158
Detach	7	JOBCTL	7-179
Directory lookup on index	15	DEVICE Read or execute access to file	7-219
Disable logins	-2	SWCTL	7-147
Disable terminal	8	HWCTL	7-192
Disk directory lookup on file name	17	DEVICE Read or execute access to file	7-224
Disk pack status	3	MOUNT HWCFG	7-160

Table 7-4: FIP SYS Calls (by Function Name) (Cont.)

Function Name	Function Code(FO)	Relevant Privileges	Page
Disk wildcard directory lookup	17	DEVICE Read or execute access to file	7-226
Drop/regain (temporary) privileges	-21	None	7-93
Enable CTRL/C trap	-7	None	7-137
Enable logins	-1	SWCTL	7-148
Enable/disable disk caching	19	TUNE	7-248
File name string scan	-10	None	7-45
File utility functions	-26	Read access to file Write access to file DATES TUNE SYSIO	7-67
Force input to terminal	-4	SYSIO	7-143
Get monitor tables - part I	-3	None	7-144
Get monitor tables - part II	-12	None	7-127
Get monitor tables - part III	-29	None	7-58
Get open channel statistics	-8	None	7-134
Hang up a dataset	-9	HWCTL	7-132
Kill job	8	JOBCTL	7-190
List logical names	21	None	7-263
List system files	23	None	7-272
Load monitor overlay code and return status	34	SWCFG	7-313
Lock/unlock job in memory	-20	TUNE	7-96

Table 7-4: FIP SYS Calls (by Function Name) (Cont.)

Function Name	Function Code(FO)	Relevant Privileges	Page
Login	4	None	7-165
Logout	5	EXQTA WACNT	7-169
Message send/receive	22	JOBCTL SEND SWCFG SWCTL SYSIO	Ch. 8
Name run-time system	-17	Write access to file	7-117
Open next disk file	33	DEVICE Read access to file Write access to file DATES	7-301
PEEK function	--	RDMEM SYSMOD	7-317
Poke memory	-6	SYSMOD	7-140
Read current privileges	28	None	7-289
Read pack attributes	-25	DEVICE	7-79
Read/read and reset accounting data	14	GACNT WACNT	7-209
Read/write account attributes	-25	GACNT WACNT	7-81
Read/write file attributes	-25	Read access to file Write access to file	7-75
Reattach	6	DEVICE	7-175
Remove logical names	21	INSTAL	7-259

Table 7-4: FIP SYS Calls (by Function Name) (Cont.)

Function Name	Function Code(FO)	Relevant Privileges	Page
Remove monitor overlay code	34	SWCFG	7-313
Remove resident library	-18	INSTAL	7-112
Remove run-time system	-18	INSTAL	7-104
Remove system files	23	Write access to file INSTAL	7-270
Return error messages	9	None	7-194
Return job status	26	JOBCTL TUNE	7-284
Send local data message with privileges	22	SEND	Ch. 8
Send Print/Batch Services request	22	None	Ch. 9
Set/clear current privileges	28	None	7-289
Set device characteristics	34	HWCFG HWCTL	7-304
Set line printer characteristics	34	HWCFG	7-307
Set system defaults	34	HWCFG SWCFG	7-310
Set number of logins	-19	SWCTL	7-98
Set password	8	GACNT WACNT	7-188
Set special run priority	-22	TUNE	7-92
Set terminal characteristics - part I	16	HWCFG	7-229
Set terminal characteristics - part II	16	HWCFG	7-241
Shut down system	-16	SHUTUP	7-119
Snap shot dump	-27	SYSIO	7-66
Special magnetic tape directory lookup	15	DEVICE	7-221

Table 7-4: FIP SYS Calls (by Function Name) (Cont.)

Function Name	Function Code(FO)	Relevant Privileges	Page
Spooling (obsolete: use PBS request)	-28	Read access to file Write access to file	7-61
Stall/Unstall system	29	HWCTL	7-291
Swap Console	6	None	7-177
Terminating file name string scan	-23	None	7-45
Third party privilege check	31	None	7-293
Unload resident library	-18	INSTAL	7-113
Unload run-time system	-18	INSTAL	7-106
Verify password	4	DEVICE GACNT WACNT	7-165
Wildcard PPN lookup	25	DEVICE	7-282
Zero a device	13	DEVICE Create/rename access to account	7-205

Cancel CTRL/O Effect on Terminal

Data Passed

Bytes	Meaning
1	CHR$(0%), the cancel CTRL/O code.
2	CHR$(N%), where N% is the number (between 0 and 12) of the channel on which the system executes the call. If you do not specify this byte, the call uses channel 0.
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 multiterminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel you specify in byte 2. If you do not specify this byte, the keyboard affected is the one open on the channel you specify in byte 2.

Data Returned

The target string is equivalent to the passed string.

Privileges Required

None.

Discussion

This call cancels the effect of a CTRL/O typed at the specified terminal. The call selects the terminal open on the channel number you pass in byte 2. (The terminal must be open on that channel.) If you use a slave terminal, byte 2 must be a nonzero channel number on which the master terminal is open; byte 3 must contain the keyboard number of the slave terminal. See the *RSTS/E System User's Guide* for a description of CTRL/O.

Enter Tape Mode on Terminal

Data Passed

Bytes	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 you do not specify this byte, the call uses channel 0.
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 multiterminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel you specify in byte 2. If you do not specify this byte, the keyboard affected is the one open on the channel you specify in byte 2.

Data Returned

The target string is equivalent to the passed string.

Privileges Required

None.

Discussion

This call is specifically for use with ASR33 terminals that have a low-speed paper tape reader. The call disables echoing on the terminal and places the terminal in tape mode so that a program can be read into the system from the low-speed reader.

The action of this call is the same as that of the TAPE command (see the *BASIC-PLUS Language Manual*). The call selects the terminal open on the channel number you pass in byte 2. (The terminal must be open on that channel.) If you use a slave terminal, byte 2 must be a nonzero channel number on which the master terminal is open; byte 3 must contain the keyboard number of the slave terminal.

Note that CTRL/C cancels tape mode.

Enable Echoing on Terminal

Data Passed

Bytes	Meaning
1	CHR$(2%), the enable echoing code.
2	CHR$(N%), where N% is the number (between 0 and 12) of the channel on which the system executes the call. If you do not specify this byte, the call uses channel 0.
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 multiterminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel you specify in byte 2. If you do not specify this byte, the keyboard affected is the one open on the channel you specify in byte 2.

Data Returned

The target string is equivalent to the passed string.

Privileges Required

None.

Discussion

This code cancels the effect of SYS calls with codes 1 and 3. The call selects the terminal open on the channel number you pass in byte 2. (The terminal must be open on that channel.) If you use a slave terminal, byte 2 must be a nonzero channel number on which the master terminal is open; byte 3 must contain the keyboard number of the slave terminal.

Disable Echoing on Terminal

Data Passed

Bytes	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 you do not specify this byte, the call uses channel 0.
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 multiterminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel you specify in byte 2. If you do not specify this byte, the keyboard affected is the one open on the channel you specify in byte 2.

Data Returned

The target string is equivalent to the passed string.

Privileges Required

None.

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 call selects the terminal open on the channel number you pass in byte 2. (The terminal must be open on that channel.) If you use a slave terminal, byte 2 must be a nonzero channel number on which the master terminal is open; byte 3 must contain the keyboard number of the slave terminal.

Note that CTRL/C reenables terminal echo.

Enable ODT Submode on Terminal

Data Passed

Bytes	Meaning
1	CHR$(4%), the enable ODT submode code.
2	CHR$(N%), where N% is the number (between 0 and 12) of the channel on which the system executes the call. If you do not specify this byte, the call uses channel 0.
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 multiterminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel you specify in byte 2. If you do not specify this byte, the keyboard affected is the one open on the channel you specify in byte 2.

Data Returned

The target string is equivalent to the passed string.

Privileges Required

None.

Discussion

ODT submode allows the system to accept less than a full line as input from the terminal. Normally, the system waits to accept terminal input until it receives a line terminated by a delimiting character: carriage return, line feed, form feed, escape character, or CTRL/D combination. However, in ODT submode the system does not wait for a delimiting character. Instead, one or more characters typed at the terminal are passed immediately to the program by the next keyboard input request statement. This input mode is called ODT submode because it is used in the system program ODT.BAS and the debugging routine ODT.OBJ.

You must enable this function before every input request statement that immediately passes characters to the program. You must use a GET statement as the input request statement. (You must not use INPUT or INPUT LINE statements, because they cause repeated generation of the input request until a line terminator is detected.)

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.

The call selects the terminal open on the channel number you pass in byte 2. (The terminal must be open on that channel.) If you use a slave terminal, byte 2 must be a nonzero channel number on which the master terminal is open; byte 3 must contain the keyboard number of the slave terminal.

Exit with No Prompt Message
F=5

Exit with No Prompt Message

Data Passed

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

Data Returned

None.

Privileges Required

None.

Discussion

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

- Keeps the files open.
- Saves the current program state, which allows you to continue execution.
- Drops temporary privilege.
- Does not generate a prompting message.
- Has the BASIC-PLUS keyboard monitor wait for a command.

FIP Function Call

The SYS call whose function code is 6 is a specialized case of the general system function call. SYS call 6 contains a subfunction code called the FIP code. The FIP code causes a dispatch call to be made to special resident or nonresident code that performs file processing. The entire class of subfunctions of SYS call 6 are called FIP calls.

See the section "SYS System Function Calls to FIP" for a description of SYS calls to the file processor.

Get Core Common String

Data Passed

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

Data Returned

The target string is the contents of the job core common area.

Privileges Required

None.

Discussion

This call 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 core common and is from 0 to 127 bytes long. This call does not alter the contents of the core common area. See SYS call 8, Put Core Common String.

Put Core Common String

Data Passed

Bytes	Meaning
1	CHR$(8%), the put string into core common code.
2-128	The string to put in core common.

Data Returned

The target string is the passed string.

Privileges Required

None.

Discussion

This call allows a program to load a single string into a common data area called core common. Another program running under the same job and called by the CHAIN statement can extract this string later. The string can be from 0 to 127 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 way to pass a limited amount of information when a program executes a CHAIN statement. If you want to pass a larger amount of information, it must be written to a disk file and read back by the later program.

Exit and Clear Program
F=9

Exit and Clear Program

Data Passed

Bytes	Meaning
1	CHR$(9%), the exit and set up NONAME code.
2-3	The first three characters of the run-time system name, in Radix-50 format, to which control is to pass. If bytes 2-5 are zero, the call selects your job keyboard monitor.
4-5	The last three characters of the run-time system name, in Radix-50 format, to which control is to pass.
6	If you do not specify this byte, the call establishes the run-time system you name in bytes 4-5 as the job keyboard monitor. Otherwise, CHR$(N%); the following values of N% determine the action performed:

Value	Action
255%	Establish the run-time system as the job keyboard monitor.
0%	Enter the specified run-time system without establishing it as the job keyboard monitor.

Data Returned

None.

Privileges Required

None.

Discussion

This call clears the current program from memory and returns control to your job keyboard monitor or the run-time system you specify 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 have the privileged

bit set in the protection code. The BASIC-PLUS command NEW NONAME performs the same action.

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 keyboard monitor. If you do not specify bytes 2 through 5, the call transfers control to the job keyboard monitor. If you specify byte 6 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 in to the system, BASIC-PLUS prints carriage return, line feed, and Ready prompt 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.

Cancel All Type Ahead

Data Passed

Bytes	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 you do not specify this byte, the call uses channel 0.
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 multiterminal service rule. The keyboard is the slave terminal under control of a master terminal open on the channel you specify in byte 2. If you do not specify this byte, the keyboard affected is the one open on the channel you specify in byte 2.

Data Returned

The target string is equivalent to the passed string.

Privileges Required

None.

Discussion

This call clears all unread, pending input from a terminal's buffers, which cancels any input typed before a program requests it. This call is mainly intended for echo control operations, where echoing of unsolicited input ruins the appearance of painted fields. See the section "Echo Control: MODE 8%" in Chapter 4 for the discussion of controlling echo and declaring a field on a screen to have a special paint character.

The call selects the terminal open on the channel number you pass in byte 2. (The terminal must be open on that channel.) If you use a slave terminal, byte 2 must be a nonzero channel number on which the master terminal is open; byte 3 must contain the keyboard number of the slave terminal.

Return Information on Last Opened File

Data Passed

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

Data Returned

Bytes	Meaning
1	The current job number times 2.
2	Internal coding.
3	The channel number (times two) on which the file was opened.
4	The most significant bits of the file size (MSB size). If the call returns a nonzero number, it indicates a file whose size is greater than 65535 blocks.
5-6+	Project-programmer number.
7-10+	File name in Radix-50 format.
11-12+	File type 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 (the same information returned by the BASIC-PLUS STATUS variable).
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 the device is part of the public structure. See Discussion.
27-30	Internal coding.

Privileges Required

None.

Discussion

When you execute a compiled program under the BASIC-PLUS run-time system (by a RUN command, a CHAIN statement, or a CCL command that executes a .BAS or .BAC file), BASIC-PLUS saves several pieces of information about the program, including its file specification and job number.

When the file is opened, BASIC-PLUS saves the information in file name string scan format (identified by the + in the Data Returned). BASIC-PLUS keeps this information until another file is opened, at which time it updates the information. This SYS call allows you to obtain the information that BASIC-PLUS saves. See the section "File Name String Scan Format" for more information.

For a file-structured OPEN, 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.

These bits are meaningless for a non-file-structured OPEN.

Examples

The following two examples illustrate the Return Information on Last Opened File SYS call:

- 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 can 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:.

- 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 that this call returns information about the file last opened, no matter how it was opened. For example, suppose the call is made after you type:

```
OLD PROG
RUN
```

The last file opened is a BASIC-PLUS work file, not the program PROG.BAS.

Execute CCL Command

Data Passed

Bytes	Meaning
1	CHR$(14%), the execute a CCL command code.
2-128	The string to be executed.

Data Returned

The target string is equivalent to the passed string.

Privileges Required

None	The protection code grants you execute access
GREAD	Execute any program within the group
WREAD	Execute any program

Possible Errors

Meaning	ERR Value
?LINE TOO LONG The string you 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 You used a nonnumeric value as an argument in one of the CCL switches. For example, a /SIZE:A switch specification can cause this error.	52
?ILLEGAL SWITCH USAGE You specified an illegal switch 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 call removes the current program from memory and executes the CCL command as though it had been typed directly to a keyboard monitor. Note that this call has the same effect on your current program as a CHAIN statement: both cause your current program to be terminated and removed from memory.

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.

Other errors can be detected after the call removes the current program and the system attempts to execute the CCL command (see the *RSTS/E System Directives Manual*).

SYS System Function Calls to FIP (F=6)

The SYS call whose function code is 6 is a specialized case of the general system function call. SYS call 6 contains a subfunction code called the FIP code. The FIP code causes a dispatch call to be made to special resident or nonresident code that performs file processing. The entire class of subfunctions of SYS call 6 are called FIP calls. Because programmers generally use FIP calls much more frequently than the SYS calls, the FIP calls are also commonly referred to as SYS calls. This chapter also uses SYS call as the preferred term.

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 subfunction code.

O$ is the optional (by function) parameter string.

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

Bytes	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 (SYS 22), the call always returns 30 bytes. Unused bytes are not defined. DIGITAL reserves the right to change the values returned in these bytes at any time.

The proper use of the FIP system function call requires that you build a parameter string to pass and that you 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. Use these descriptions to determine whether you need the information.

Building a Parameter String

Some SYS calls require no parameters except the function and subfunction codes; other SYS calls require either variable length parameter strings or very simple parameter strings. For such SYS 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 CTRLC TRAP
        !(NO PARAMETER STRING)

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

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

Many SYS calls require more complex data formats. For example, the Kill A Job SYS call, (SYS 8), requires byte 3 to be the job number to kill, byte 27 to be 0, and byte 28 to be 255. To build the complex parameter string to pass to a function, DIGITAL recommends that you dimension a 30-element integer array and set the items in the array to values that map into those required in the parameter string format. You can then convert the array to a character string by the CHANGE statement before passing it as the parameter string of the SYS system function call. The resulting character string is in the proper format and contains the correct byte values to be placed as the parameter string of the SYS 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 that builds the list is the CHANGE statement and the call itself:

```
100   CHANGE A% TO A$              !GENERATES CHARACTER
        .                          !STRING FROM THE
        .                          !INTEGER LIST
        .
200   B$ = SYS(A$)                 !INVOKE SYSTEM FUNCTION CALL
```

In the SYS call descriptions, certain parts of parameter strings are documented as "Reserved; should be 0." You should fill these bytes with NUL characters (ASCII code 0). You can use the STRING$(n,0%) function (where n is the number of NUL characters needed) to generate a string of proper length or place 0% in the appropriate array elements. By placing 0% in these bytes you will be sure that your code is upward compatible if future releases of RSTS/E use these currently unused bytes. If not, your code may produce unpredictable results with future releases of RSTS/E.

Unpacking the Returned Data

In the example shown in the previous section, the action performed (kill a job), rather than the data returned, is the objective of the call. However, many SYS calls return a data string that is your primary objective. In such a case, you must unpack the data in the string.

When you build the parameter string, DIGITAL recommends two ways to unpack the returned string:

Method 1:

If you need only a few pieces of data, it may be more convenient to operate directly on the returned string. For example, if you want only the 4-byte Radix-50 representation of a 6-byte string, you can use the File Name String Scan SYS call (SYS -10):

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

The MID function extracts bytes 7 through 10 of the returned string. To extract numeric data, you can use the ASCII or CVT$% functions. See the *BASIC-PLUS Language Manual* for more information.

Method 2:

If you need many pieces of the returned data, or if you need to use the string returned by the SYS call to set up another SYS call, you can transform the returned string to a 30-element integer array using a CHANGE statement. For example:

```
CHANGE A$ TO A%

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

When you convert the returned string in this manner, you need to do further conversions to get numeric data into a usable form. Consider, for example, the data returned by a the Directory Lookup On Index call (SYS 15). The layout of the data returned specifies that bytes 11 and 12 are the file type encoded in Radix-50 format. To convert those bytes into an ASCII string (for example, to open the file), you must convert the two bytes to a single integer and then use the BASIC-PLUS RAD$ function. However, the integer representation of each byte occupies a full word; 16 bits in length.

Figure 7-1 shows array elements 11 and 12.

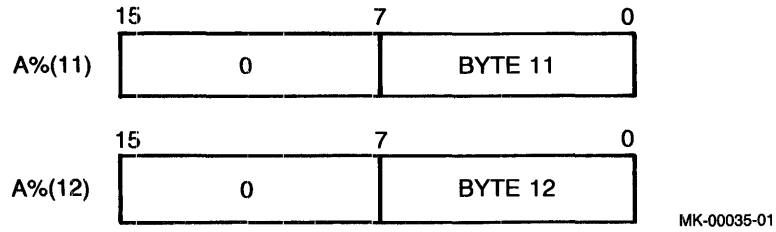


Figure 7-1: Integer Representation of Changed Characters

A%(11) contains the low byte portion of the Radix-50 word; A%(12) contains the high byte portion of the Radix-50 word. You must combine the two bytes into a single word and convert them to the proper character string representation:

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

Figure 7-2 shows that the SWAP% function reverses the bytes (the low byte takes the high byte position and vice versa) in an integer word.

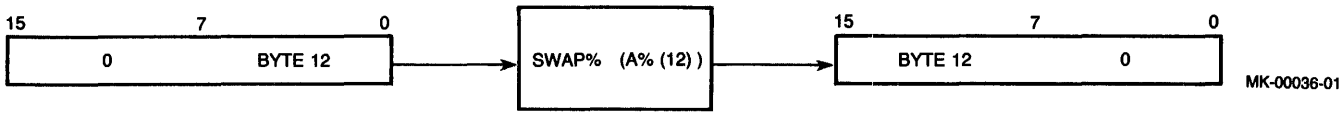


Figure 7-2: Reversal of Bytes by SWAP%() Function

Thus, byte 12 takes the high byte position in the word. The + operator then combines the two words to form one word. The RAD$ function performs the conversion on that one integer word to produce the three-character string representation of the file type. See 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, you must use this procedure on each pair of bytes in the string. For example, SYS call 15 returns the file name in bytes 7 through 10. 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$
```

You can also use the statement:

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

Notation and References Used in SYS Call Descriptions

This section describes conventions used in the SYS call descriptions. It also provides programming hints for working with SYS calls. Because programmers commonly refer to the FIP calls as SYS calls, the term SYS call is used in the individual description of each call.

Project-Programmer Number

Many SYS calls require that you specify a project-programmer number (PPN) in the calling string, and several return a PPN. In these cases, the PPN field is in the general form:

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

where:

```
Byte X       holds the programmer number

Byte (X+1)   holds the project number.
```

For example, to set up a SYS call to zero an account on a disk (SYS 13), the calling format shows:

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

If the call is to be set up in a 30-element array A%, then the format requires that:

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

Integer (2-Byte) Numbers

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

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

If you are processing the returned string directly (that is, without changing it to an integer array), then you can obtain the integer value of the two bytes with the statement:

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

where A$ holds the returned string. See the *BASIC-PLUS Language Manual* for a discussion of the SWAP% function with the CVT functions.

If you convert the returned data string to an integer array A% using the CHANGE statement, then you can obtain the integer value with the statement:

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

For example, the Get Monitor Tables - Part I SYS call (SYS -3) returns the address of the 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%)))

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

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, the call returns an unsigned integer 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 following statement places the full 16-bit value into the floating-point variable Q:

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

where 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.

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

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

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

```
10  Q% = Q - 32768.
         \ Q% = Q% EQV 32767%
         \ A%(5%) = Q% AND 255%
         \ A%(6%) = SWAP% (Q%) AND 255%
```

However, this second method requires more code.

Negative Byte Values

Many FIP calls pass and return integer values in one byte of the data string. Some call descriptions refer to negative byte values.

While negative byte values are meaningful to MACRO programmers, BASIC-PLUS treats all byte values as positive. Where the term "negative" byte value is used, it refers to an integer value between 128% and 255%. To obtain the actual signed value, use the following statement:

```
S% = SWAP%(B%)/256%
```

where B% is the byte to convert.

File Name String Scan Format

The File Name String Scan SYS call (SYS -10) is useful as a "front-end" for many SYS functions. Most of the SYS calls that require device or file information in their parameter strings expect information in the format in which the SYS -10 call returns it. For example, SYS call 17, Disk Directory Look Up On File Name, expects its calling string to be passed in exactly the same format as that returned by the SYS -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 File Name String Scan SYS 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 file name, password, pack identification label or other six-character string to be passed as two words in Radix-50 format. The File Name String Scan call is the only means provided to convert the string to the proper format. The section "File Name String Scan" (SYS=-10, SYS=-23) describes how this conversion is done.

Note

The SYS call descriptions that follow use a special convention to avoid repetition. A plus sign (+) postscript identifies fields in the calls that are either passed or returned in the same format as that returned by SYS call -10, File Name String Scan. See the section "File Name String Scan" (SYS=-10, SYS=-23) for a detailed description of the fields returned by File Name String Scan.

See Table 7-3 in the beginning of this chapter for a quick reference index of the SYS functions ordered by FIP code (F0). See Table 7-4 for a quick reference index of the SYS functions arranged alphabetically by function name.

MACRO Mnemonic Cross-References

The *RSTS/E System Directives Manual* describes monitor directives for MACRO programmers. Many directives correspond to the SYS calls described in the following sections. In each section that follows, the SYS call number (F0 =) appears in bold type at the top of the page. The corresponding MACRO directive appears in parenthesis below it. For a summary of SYS call codes and their corresponding monitor directives, see Table F-1. For information on the use of MACRO directives, see the *RSTS/E System Directives Manual*.

Organization of This Section

The system function calls to FIP are listed by number, from the most negative to the most positive. There are three exceptions to this sequence:

- File Name String Scan (F0=-10, F0=-23). Because this call is used as a "front end" for many calls, it is described first.
- Directory Lookup Calls (F0=15, F0=17). Because these calls are related, SYS 17 is right after SYS 15.
- Message Send/Receive (F0=22). See Chapters 8 and 9 for a description of this call.

The PEEK function is described at the end of this chapter.

File Name String Scan

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-10), the file name 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:

Bytes	Meaning
1	The current job number times 2.
2	The Most Significant Bits (MSB) of the file size as specified in the /FILESIZE:n (or /SIZE:n) file specification switch. If the call returns a nonzero number, it indicates a file whose size is greater than 65535 blocks.
3-4	Internal coding.
5-6	Project-programmer number (PPN). 0 means the current account. See the Discussion for information about translation of special characters.
7-10	File name in Radix-50 format. See Discussion.
11-12	File type 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 Least Significant Bits (LSB) 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; 0 if /MODE or /RONLY were not specified.

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 job default protection is currently assigned. If a protection code is found or if no protection code is found when a job default protection is currently set, this byte is nonzero and byte 22 contains the protection code.

22 Protection code when byte 21 is nonzero.

23-24 To determine what is returned for a device, flag word 2 must be checked. If no colon was found in the string, these two bytes and byte 25 and 26 are 0. If a colon was found, a device name may or may not have been found.

A device name can be a physical device name or a logical device name. If a physical device name was found, these bytes contain two characters in ASCII format. (For example, DK yields D in byte 23 and K in byte 24.) Bytes 25 and 26 contain unit number information. If a logical name (either job-specific or system-wide) was found and that logical name was translatable (the name was currently assigned to a physical device), the call translates the name and returns the full physical device information in bytes 23 through 26. If the logical device name was untranslatable, the call returns the logical name in Radix-50 format in bytes 23 through 26. For logical names longer than 6 characters, the call returns only the first 6 characters. 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.

Privileges Required

None.

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME The character string scanned contains unacceptable characters. See the *RSTS/E System User's Guide* for a description of a file specification. If you are using the -10 version of the call, 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 file name string scan function determines specific file syntax information (for example, whether a given file name is valid) and returns information in the format required for all other file- and device-related SYS calls. The call also processes the allowable RSTS/E file specification switches. See the *RSTS/E System User's Guide* for a description of the format of these switches.

Note

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

The call does the following for each component of a file specification:

- For a device specification, the call processes physical device names and unit number information. If you pass a logical name, 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 call does not translate the logical name. The STATUS variable is set for the device type found in the string scanned.

- For a project-programmer specification, the call validates the format. If you pass a character denoting an account, the call translates it to the proper numbers. For example, if $ is assigned to the system library account, [1,2], $ is returned as 2 in byte 5 and 1 in byte 6. Besides the $, the call also translates the characters !, %, &, # and @ if they are assigned to accounts and indicates whether the wildcard character was found.

Note

Special PPN characters other than the dollar sign ($) may not be available in future releases of RSTS/E.

- For a file name, the call validates the format and translates the name into Radix-50 format. It also notes the presence of wildcard characters.

- For a file type, the call validates the format and translates it into Radix-50 format. The call also notes the presence of wildcard characters.

- 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.

- 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 shows how to convert a string to Radix-50 format with a user-defined function and the file name string scan SYS call:

```
10  DEF FNP0$(A$) = MID (SYS(CHR$(6%)+CHR$(-10%)+A$),7%,4%)&
\   ! PACK 6 CHARACTERS TO RADIX-50
```

The function FNP0$ returns a four-character string that is the Radix-50 representation of the first six characters of A$. (Note that the function does not include error handling and that errors can occur.) The File Name String Scan SYS call is the only function that packs a string in Radix-50 format. To pack strings longer than six characters, you must make multiple calls to the SYS function. You can pack up to nine characters in a single call if a period separates the first six characters from the last three characters (the file name and type 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 you create these two words, 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 holds a YES or NO answer; see Tables 7-5 and 7-6 for details.

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

Tables 7-5 and 7-6 assume 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 7-5: File Name String Scan Flag Word 1

Flag word 1: where S0% = M%(27%)+SWAP%(M%(28%))		
Bit	**Comparison**	**Meaning**
0	(S0% AND 1%)<>0%	The /CLUSTERSIZE:n switch was specified.
	(S0% AND 1%) = 0%	No /CLUSTERSIZE:n was found.
1	(S0% AND 2%)<>0%	Either the /MODE:n or /RONLY switch was specified.
	(S0% AND 2%) = 0%	Neither /MODE:n nor /RONLY was found.

Table 7-5: File Name String Scan Flag Word 1 (Cont.)

Bit	Comparison	Meaning
2	(S0% AND 4%)<>0%	Either the /FILESIZE:n or /SIZE:n switch was specified.
	(S0% AND 4%) = 0%	Neither the /FILESIZE:n nor /SIZE:n switch was found.
3	(S0% AND 8%)<>0%	The /POSITION:n switch was specified.
	(S0% AND 8%) = 0%	No /POSITION:n switch was found.
4-7	Reserved.	
8	(S0% AND 256%)<>0%	A file name was found in the source string (and is returned in Radix-50 format in bytes 7 through 10).
	(S0% AND 256%) = 0%	No file name was found.
9	(S0% AND 512%)<>0%	A period (.) was found in source string.
	(S0% AND 512%) = 0%	No period was found in source string implying that no file type was specified.
10	(S0% AND 1024%)<>0%	A project-programmer number (PPN) was found in source string.
	(S0% AND 1024%) = 0%	No PPN was found.
11	(S0% AND 2048%)<>0%	A left angle bracket (<) or /PR was found in source string, implying that a protection code was found.
	(S0% AND 2048%) = 0%	No left angle bracket (<) or /PR was found (no protection was specified).
12	(S0% AND 4096%)<>0%	A colon (but not necessarily a device name) was found.
	(S0% AND 4096%) = 0%	No colon was found, implying that no device could have been specified.
13	(S0% AND 8192%)<>0%	Device name was specified and was a logical device name.
	(S0% AND 8192%) = 0%	Device name (if specified) was an absolute (nonlogical) device name. (If device name was not specified, this is 0.)

Table 7-5: File Name String Scan Flag Word 1 (Cont.)

Bit	Comparison	Meaning
15	S0%<0%	Source string contained wildcard characters (either ? or * or both) in file name, file type or PPN 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. You must test bits of flag word 2 for wildcard characters and device name found.

Table 7-6: File Name String Scan Flag Word 2

Flag Word 2: where S1% = M%(29%)+SWAP%(M%(30%))		
Bit	**Comparison**	**Meaning**
0	(S1% AND 1%)<>0%	File name was found in the source string.
	(S1% AND 1%) = 0%	No file name was found. The next two comparisons return 0.
1	(S1% AND 2%)<>0%	File name was an asterisk (*) character and is returned in bytes 7 through 10 as the Radix-50 representation of the string "??????".
	(S1% AND 2%) = 0%	File name was not an * character.
2	(S1% AND 4%)<>0%	File name contained at least one question mark (?) character.
	(S1% AND 4%) = 0%	File name did not contain any ? characters.
3	(S1% AND 8%)<>0%	A period (.) was found.
	(S1% AND 8%) = 0%	No period was found, implying that no file type was specified. The following three comparisons return 0.

Table 7-6: File Name String Scan Flag Word 2 (Cont.)

Bit	Comparison	Meaning
4	(S1% AND 16%)<>0%	A file type was found (that is, the field after the period was not null).
	(S1% AND 16%) = 0%	No file type was found. (The field after the period was null -- the next two comparisons return 0.)
5	(S1% AND 32%)<>0%	File type was an * character and is returned in bytes 11 and 12 as the Radix-50 representation of the string "???".
	(S1% AND 32%) = 0%	File type was not an * character.
6	(S1% AND 64%)<>0%	File type contained at least one ? character.
	(S1% AND 64%) = 0%	File type did not contain any ? characters.
7	(S1% AND 128%)<>0%	A PPN number was found.
	(S1% AND 128%) = 0%	No PPN was found. (The next two comparisons return 0.)
8*	(S1% AND 256%)<>0%	Project number was an * character (that is, the PPN 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 PPN 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.

Table 7-6: File Name String Scan Flag Word 2 (Cont.)

Bit	Comparison	Meaning
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 logical device name specified was invalid for one of the following reasons: o The device name contained an underscore (_) but did not correspond to any physical device on the system. o The device name did not contain an underscore but could not be translated to a physical device name. 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 PPN was of the form [*,*], then both bit 8 and bit 9 of the data byte returned are nonzero 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 sample program uses this word and prints out a list of all the bits returned in the word.

```
5       DIM M%(30%)             ! SET UP AN ARRAY TO RETURN TO
10      PRINT "STRING TO SCAN";
20      INPUT LINE S$
30      S$=CVT$$(S$,-1%)        ! GET RID OF GARBAGE BYTES
40      CHANGE SYS(CHR$(6%)+CHR$(-10)+S$) TO M%
50      S1%=M%(29%)+SWAP%(M%(30%))
100     IF S1% AND 1%     THEN PRINT "FILENAME FOUND"
110     IF S1% AND 2%     THEN PRINT "FILENAME WAS AN '*'"
120     IF S1% AND 4%     THEN PRINT "FILENAME HAD '?'S"
130     IF S1% AND 8%     THEN PRINT "DOT (.) FOUND"
140     IF S1% AND 16%    THEN PRINT "NON-NULL FILE TYPE FOUND"
150     IF S1% AND 32%    THEN PRINT "FILE TYPE WAS '*'"
160     IF S1% AND 64%    THEN PRINT "FILE TYPE 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 previous messages:

```
STRING TO SCAN?  ABCDEF.TYP
FILENAME FOUND
DOT (.) FOUND
NON-NULL FILE TYPE FOUND

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

STRING TO SCAN?  SY:FILENM.TYP[1,203]
FILENAME FOUND
DOT (.) FOUND
```

```
NON-NULL FILE TYPE FOUND
PPN FOUND
COLON (:) FOUND
DEVICE NAME FOUND
'STATUS' HAS BEEN SET

STRING TO SCAN?  SY:FILENM.TYP[2,103]/PR:52
FILENAME FOUND
DOT (.) FOUND
NON-NULL FILE TYPE FOUND
PPN FOUND
PROTECTION CODE FOUND
COLON (:) FOUND
DEVICE NAME FOUND
'STATUS' HAS BEEN SET

STRING TO SCAN?  SY:FILENM.TYP[,201]
FILENAME FOUND
DOT (.) FOUND
NON-NULL FILE TYPE 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 FILE TYPE FOUND
FILE TYPE WAS ''
COLON (:) FOUND
DEVICE NAME FOUND
'STATUS' HAS BEEN SET

STRING TO SCAN?  SY:FILE??.TYP
FILENAME FOUND
FILENAME HAD '?'S
DOT (.) FOUND
NON-NULL FILE TYPE 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:

- When no device name is found in the string; that is, no colon is found, the STATUS is unpredictable. This condition applies when bit 12 of flag word 2 tests as equal to 0.

- 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 unpredictable. 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%).

- 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 file name string scan call has 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, it generates the error ?Illegal file name (ERR=2). The -23 version of the call processes a full command line, which can contain multiple file specifications and switches other than valid forms of the file specification switches. To process a full command line, the call terminates the scan on certain characters.

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

=	Equal sign
/	Slash unless part of a valid file specification switch
;	Semicolon
,	Comma unless between brackets or parentheses (indicates PPN) end of string

The scan is done from left to right. If the scan finds a valid file specification switch, it processed the switch and continues the scan. If the scan finds other than a file specification switch, the scan terminates. The program must process the switch and also check for remaining switches. The scan does not process any file specification switches following a switch that terminates the scan.

The BASIC-PLUS variable RECOUNT returns the number of unscanned characters. For example:

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

This call returns the data as described for CHR$(-10%) and RECOUNT equals 0. The following call returns the data described for CHR$(-10%) for the string "SY:[1,4]ABC/PR:40" and RECOUNT equals 7:

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

The scan terminates on the comma between file specifications. Any other characters generate an error and none of the data is returned.

Get Monitor Tables - Part III

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-29%), the get monitor tables - part III code.
3-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1	The current job number times 2.
2	Not used.
3-4	(DDCTBL) - The controller/device table.
5-6	(UCTTBL) - The unit/controller table.
7-8	(SATEND) - The disk size table.
9-10	(UNTLVL) - The disk structure level table.
11-12	(MFDPTR) - The MFD pointer table.
13-14	(MAGLBL) - The magnetic tape label default table.
15	The number of jobs currently on the system.
16-20	Internal code.
21-22	Hardware configuration word. See Discussion.
23-24	(UNTERR) - The unit error table.
25-26	(DEVCLU) - The low byte contains the device cluster size. The high byte contains the CLUFAC table.
27-28	(NULRTS) - The null run-time system block pointer.
29-30	(DSTPTR) - The memory management unit (MMU) address of the disk statistics table if this monitor is generated with the unsupported disk statistics feature. Otherwise, 0.

Privileges Required

None.

Possible Errors

None.

Discussion

The three Get Monitor Table SYS calls to FIP return to your program either an address or a data value. The calls are commonly used with the PEEK function to read various system parameters and tables that 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. For a description of Get Monitor Tables - Part I, see SYS call -3. For a description of Get Monitor Tables - Part II, see SYS call -12. The section "The PEEK Function" describes the use of the PEEK function for certain convenient programming operations.

In this call, a name in all uppercase letters denotes each item of information described. This name is the same one used to identify the information in the RSTS/E assembly listings. If the name is in parentheses, the information returned is an address of the data described. If the name is not in parentheses, the information returned is the actual data value. For example, Get Monitor Tables - Part I returns CNT.KB-1 in byte 3. The value returned is the number of terminal lines minus 1 configured on the system. However, bytes 11 and 12 return (JOBTBL), the address of the table of jobs. Use the PEEK function to inspect the address.

Note

> All information returned by the call described in this section is internal to RSTS/E and is subject to change at any time.

DDCTBL and UCTTBL (bytes 3-6) are pointers to monitor tables that allow system programs to translate communications device names from one format to another. For example, DECnet, which runs on many different DIGITAL systems, uses a different format for device names than RSTS/E. Thus, programs that print information about communications devices need these tables. The SYSTAT system program also uses this call to print its busy devices and disk status reports.

SATEND (bytes 7-9) is a pointer to a disk size table that the SYSTAT program uses to compute sizes for its display. UNTLVL (bytes 9-10) is

a pointer to the disk structure level table, MFDPTR (bytes 11-12) is a pointer to the MFD pointer table, and MAGLBL is a pointer to the magnetic tape label table. Byte 15 contains the number of entries in the monitor's job table structure that includes jobs in any state.

The hardware configuration word (bytes 21-22) contains a bit mask specifying configuration data. The most useful bit flags are the following:

Value	Meaning
32%	Q-BUS system. If bit is OFF, UNIBUS system.
512%	FPP available. If bit is OFF, FPP is not available.
1024%	CIS available. If bit is OFF, CIS is not available.
8192%	System has Instruction and Data (I&D) space. If bit is OFF, system does not have I&D space.

Spooling

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-28%), the spool request code.
3-4	Reserved; should be 0.
5-6+	The PPN of the file to spool. If bytes 5-6 are zero, the call uses the current user account. The call does not allow wildcards.
7-10+	The file name (which can include wildcards), in Radix-50 format, of the file to spool.
11-12+	The file type (which can include wildcards), in Radix-50 format, of the file to spool.
13-14	The two-character, ASCII spooled device name field to which the file is sent. If bytes 13-14 are zero, LP is used. These bytes may affect whether requests are channeled to the Print/Batch Services (PBS) package or the OPSER-based spooling package. See Discussion.
15	The unit number of the device name field specified in bytes 13-14.
16	The unit number real flag of the device specified in bytes 13-14. Specify -1 if byte 15 contains an actual unit number. Specify 0 if bytes 13-14 contain a generic device name, in which case the monitor issues a request for the default print or batch queue. See Discussion.
17-18	Reserved, must be zero.
19-20	The flag word to specify whether to route the request to the Print/Batch Services (PBS) or the OPSER-based (OPSER) spooling package:

Value	Meaning
0%	The default. See Discussion.
4096%	Network print or batch request. Only meaningful for PBS. See Discussion.

8192% Always route request to the OPSER-based spooling package

16384% Always route request to the PBS package

You can specify the following values for the PBS package:

Value	Meaning
4%	Delete the file after spooling; same as DCL PRINT command's /DELETE qualifier.
32%	No header; same as DCL PRINT command's /NOFLAG_PAGES qualifier. This value is ignored for batch requests.

You can specify the following values for the OPSER spooling package:

Value	Meaning
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 or DCL PRINT command's /DELETE qualifier.
8%	Binary file; equivalent to QUE /BI option.
16%	End; equivalent to QUE /END option.
32%	No header; equivalent to QUE /NH option or DCL PRINT command's /NOFLAG_PAGES qualifier.

21-22 Reserved; should be 0.

23-24+ The device name where the file to be spooled is located. 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. This byte is ignored if byte 26 is zero.

26+ The unit number real flag of the device containing the file to be spooled. A nonzero value indicates a real unit number in byte 25.

27-30 Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

None	Spool a file if the protection code permits access
GREAD	Spool a file /NODELETE in any account within the group
WREAD	Spool any file /NODELETE
GWRITE	Spool a file /DELETE in any account within the group
WWRITE	Spool any file /DELETE

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 file name or type specified in bytes 7-12 cannot be found, or neither PBS nor QUEMAN (OPSER spooler) is 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 or queue a compiled file.	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	21

mount the disk with the MOUNT command (requires MOUNT privilege).

?DISK PACK IS LOCKED OUT 22
The disk is in a locked state. Execute the call under a sufficiently privileged account to override this condition.

?DEVICE NOT FILE STRUCTURED 30
The device specified in bytes 23-24 of the call is not a file-structured device.

?NO BUFFER SPACE AVAILABLE 32
System buffers are not currently available to store this message. This may be a transient condition; retry the operation.

Discussion

RSTS/E has two spooling packages: the Print/Batch Services package (PBS) and the OPSER-based spooling package (OPSER).

The system sends the request either to PBS or OPSER according to the value you specify in bytes 19-20. Bits 8192% and 16384%, if set, determine the routing. If both bits are clear, then the next two rules apply:

- If the spooled device name field in bytes 13-14 is null or LP, then the system sends the request to PBS if it is running. Otherwise, the system sends the request to OPSER.
- If the spooled device name field in bytes 13-14 is BA and the filetype is .COM, then the request is routed to PBS. Otherwise, the system sends the request to OPSER.

PBS and OPSER interpret the device name field passed in bytes 13-14 and 15 differently.

PBS requests:

Data Passed	Call Interpretation
null	Default print queue
LP:	Default print queue
LPn:	Print queue named LPn:
BA:	Default batch queue
BAn:	Batch queue named BAn:

If you specify the value 4096% in bytes 13-14, PBS sends print requests to NET$PRINT and batch requests to NET$BATCH.

OPSER requests:

Data Passed	Call Interpretation
null	Print queue LP0:
LP:	Print queue LP:
LPn:	Print queue LPn:
BA:	Batch queue BA:
BAn:	Batch queue BAn:

Byte 16 is the unit number real flag. A nonzero value instructs the monitor to issue the request for the queue with the same name as the device name field. For this to work properly with the PBS package, the system manager must define queues named LP0: - LP7:, and BA0: - BA7:.

When the monitor executes this call, it performs the following checks:

1. Ensures that the specified file name 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 access to it.

3. If no wildcards are 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 the spooler is available (not hibernating).

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

You should use the Send User Request SYS call in future applications. See Chapter 9, "System Call for Print/Batch Services."

Snap Shot Dump

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-27%), the snap shot dump code.
3-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

SYSIO

Possible Errors

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT The call attempted to write data to the crash dump file, but crash dump was not enabled at system start-up time because sufficient space was not available on the system disk.	5

Note that this call also returns device-dependent errors such as ?Device hung or write locked (ERR=14).

Discussion

This call writes the current monitor image executing in memory and the contents of the extended buffer pool (XBUF) to the crash dump file [0,1]CRASH.SYS. XBUF contains monitor data structures, including DECnet/E data structures and caching information. You can analyze the contents of the CRASH.SYS file with the ANALYS program (see the *RSTS/E System Manager's Guide*).

File Utility Functions

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-26%), the file utility code.
3	CHR$(N%), where N% is the internal channel number (in the range 1 to 12) on which the file is open. If N% is 0, specify the target file by PPN and file name and type in bytes 5 through 12.
4	The first flag byte. (Byte 27 is the second flag byte). CHR$(F%), where F% specifies the file utility function. The function F% is one (or the sum) of the following codes:

Value	Meaning
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. Requires DATES privilege if the date of last access is changed (bytes 17-18 are nonzero).
8%	Change the file's run-time system name field.
16%	Return the file's retrieval information. That is, 16% causes the monitor to map the virtual block number (VBN) of the file into the disk DCN. You can use this code 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 noncontiguous.

64% Enable/disable sequential mode caching if the file is cached. You cannot use this with code 8%. Also see bytes 13-16. (Requires TUNE privilege.)

128% Enable/disable data caching on the file. You cannot use this with code 8%. See byte 15. (Requires TUNE privilege.)

5-6+ If N% in byte 3 is 0, specify the PPN of the file you want to modify.

If N% is nonzero, these bytes are ignored.

7-10+ If N% in byte 3 is 0, specify the file name (in Radix-50 format) of the file you want to modify.
If N% is nonzero, the call ignores these bytes.

11-12+ If N% in byte 3 is 0, specify the file type (in Radix-50 format) of the file you want to modify.
If N% is nonzero, the call ignores these bytes.

13-16 The specifications in these bytes depend on the function code specified in byte 4:

If byte 4 AND 8%<>0%, then bytes 13 through 16 contain the new run-time system name field in Radix-50 format.

If byte 4 AND 16%<>0%, then bytes 13 and 14 contain the low order word of the VBN you want to locate, byte 15 contains 0% or is used by another operation, and byte 16 contains the high order byte of the VBN you want 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:

Flag	Meaning
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%.
32%	New value for no delete/rename bit if byte 27 AND 1%<>0%.
128%	New value for cached bit if byte 4 AND 128%<>0%.

17-18 If you select the change file backup statistics function in byte 4 (code 4), these bytes specify a new date of last access for the file. If you do not want to change the date, specify 0. If you do not select the statistics function, the call ignores these bytes.

19-20 If you select the change file backup statistics function in byte 4 (code 4), these bytes specify a new date of creation for the file. If you do not want to change the date, specify 0. If you do not select the statistics function, the call ignores these bytes.

21-22 If you select the change file backup statistics function in byte 4 (code 4), these bytes specify a new time of creation for the file. If you do not want to change the time, specify 0. If you do not select the statistics function, the call ignores these bytes.

23-24+ If N% in byte 3 is 0, specify the name of the device that contains the file you want 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 nonzero, the call ignores these bytes.

25-26+ If N% in byte 3 is 0, specify the unit number and unit number flag associated with the file you want to modify.

If N% is nonzero, the call ignores these bytes.

27 The second flag byte. CHR$(K%), where K% is the sum of the selected functions:

Value	Meaning
1%	Change the value of the file's no delete/rename bit. See byte 15. Cannot be used with byte 4, code 8%. (Requires SYSIO privilege.)
2%	Do not return the error ?Protection violation if the operation will not succeed. See Discussion.

28-30 Reserved; should be 0.

Data Returned

Bytes	Meaning
1	Not used.
2	The file characteristics: 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=32% File has the no delete/rename bit set. 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 zeros.
27-30	The file's run-time system name in Radix-50 format.

Privileges Required

None	Read or set file flags, if the protection code permits
GREAD	Read file flags in any account within the group
WREAD	Read file flags in any account
GWRITE	Set file flags in any account within the group
WWRITE	Set file flags in any account
DATES	Change file last access date
TUNE	Set or clear file caching bits
SYSIO	Set or clear nodelete/rename bit

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 Name Run-time System SYS call (SYS -17) and the Change File Backup Statistics SYS call (SYS -11). This call provides support for files larger than 65535 blocks and 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 (UFD) contains file size information for large files. The call decodes the two-word run-time system name field as follows:

- If the first word is nonzero, the data in both words is the run-time system name. The file size is limited to 65535 blocks.
- 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:

- Because an executable file cannot have both a run-time system name and a most significant bit indication in the field, large files are not executable.

- You cannot extend a compiled file beyond block 65535. An attempt to extend a compiled file past block 65535 results in the error ?Protection violation (ERR=10).

- 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.

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

- You cannot change the run-time system name of a file that is larger than 65535 blocks. The attempt results in the error ?Protection violation (ERR=10).

- You cannot change the run-time system name of a compiled file to two words of zeros. The attempt results in the error ?Protection violation (ERR=10). Note that you can perform this operation on a noncompiled file.

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

To place a file in a particular position on the disk, 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). 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.

Note that you can use either this call or SYS call -11, Change File Statistics, to change data in a file's accounting entry. However, the two calls work differently when you open a file, write to it, change the date of last access in the file's accounting entry, and then close the file.

When you use this SYS call to change the date of last access before closing the file (by specifying 4% in byte 4 and a new date in bytes 17 and 18), the system updates the file's accounting entry to contain the current date when it closes the file. Use SYS call -11 if you want the file's accounting entry to retain the date specified in the call after the file is closed.

To change the value of the file's no delete/rename bit, pass the new value as the value 32% in byte 15. The contents of the file's no delete/rename bit is returned as the value 32% in byte 2. An attempt

to reset the bit in the following files generates the error ?Protection violation: [0,1]SATT.SYS, [0,1]BADB.SYS, or SY0:[0,1]INIT.SYS. For more information about the no delete/rename bit, see the REFRESH FILE suboption of INIT, in the *RSTS/E System Installation and Update Guide*.

Because you can specify several functions for this call to perform at once, you can use the value 2% in byte 27 to avoid the error ?Protection violation. This value instructs the call to disregard any invalid requests while still processing valid ones. In previous versions of RSTS/E, the call returned the error if any requested function could not be executed, even if some of the requested functions were perfectly valid.

Manipulate File, Pack, and Account Attributes

This call has the following subfunctions:

- Read File Attributes
- Write File Attributes
- Read Pack Attributes
- Read Account Attributes
- Write Account Attributes
- Delete Account Attributes

Certain PDP-11 record organizations, such as RMS-11, define characteristics for files that they create. These characteristics are called file attributes. File attributes are defined when the file is created and must be retained during the existence of the file. In RSTS/E, file attributes are kept on disk in a UFD entry. See the *RSTS/E System User's Guide* for a description of file attributes.

Account attributes, on the other hand, are divided into "attribute blocks." Each block is identified by a type code in the range 1 to 255 and contains 13 bytes of data. The account attribute calls identify the attribute to be accessed using the type code. Type codes in the range 1 to 127 are reserved for use by DIGITAL. Type codes in the range 128 to 255 are for customer use. Customer applications (typically account management related programs) can use these codes for storing moderate amounts of account-related information. Because excess use of account attributes decreases the number of possible accounts per group, applications that need to store a lot of data should use an auxiliary file. Currently, approximately five additional (user-supplied) account attributes can be used per account without affecting the 255 account per group limit. Note that this is subject to change because future releases of RSTS/E may use additional account attributes.

Because these subfunctions deal with internal data structures, any reading or writing account attributes controlled by DIGITAL may cause problems in future releases. If you have data that needs to be manipulated or read by a significant number of programs, use one of the other SYS calls provided.

The account attribute calls differ from the file attribute calls by the negative value passed in byte 3 rather than a channel number of an open file.

Read File Attributes

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-25%), the read/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	Reserved; should be 0.

Data Returned

Bytes	Meaning
1	Current job number times 2.
2-4	Not used.
5-26	File attribute data. If file has no attributes, all bytes contain nulls*.
27-30	Name of run-time system under which file was created, in Radix-50 format.

* 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.

Privileges Required

None.

Read File Attributes
F0=-25

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, or the channel is open on a UFD. (UFDs do not have attributes.)	10
?DEVICE NOT FILE STRUCTURED Device on which file is open must be disk.	30
?ILLEGAL I/O CHANNEL Attributes can be accessed only on channels 1 through 15.	46

Write File Attributes

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-25%), the read/write attributes code.
3	CHR$(N%), where N% is the channel number on which file is open. (You must have write access on the open channel.)
4	CHR$(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	Reserved; should be 0.

Data Returned

None.

Privileges Required

None.

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, or the channel is open on a UFD. (UFDs do not have attributes.)	10
?DEVICE NOT FILE STRUCTURED Device on which file is open must be disk.	30

?ILLEGAL BYTE COUNT FOR I/O 31
No more than 11 can be specified in byte 4.

?ILLEGAL I/O CHANNEL 46
Attributes can be accessed only on channels 1 through 15.

Note

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

Read Pack Attributes

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-25%), the read/write attributes code.
3	CHR$(-4%), the code to read pack attributes.
4-22	Reserved; should be 0.
23-26+	The name and unit number of the disk device whose attributes are to be returned.
27-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1-6	Not used.
7-8	Starting device cluster number of the MFD.
9-10	Pack revision level.
11	Pack cluster size.
12	Not used.
13-14	Pack status/flags. See the Discussion.
15-18	Pack ID, in Radix-50 format.
19-20	Size of disk in device cluster numbers.
21	Device cluster size.
22	0 if disk is not system disk; 1 if disk is system disk.
23-24	UNTCNT for the disk.

25-26 Reserved for special applications.

27-28 Number of free device clusters.

29-30 Not used.

Privileges Required

DEVICE Access a restricted disk

Possible Errors

Meaning	ERR Value
?NOT A VALID DEVICE Device specified is not a valid device.	6
?DEVICE NOT FILE STRUCTURED Device specified is not a logically mounted disk.	30

Discussion

This call returns information about mounted disks. You can use it to obtain the characteristics of a disk and the drive on which the disk is mounted.

The following are the defined bits returned in the pack status/flags:

Bit	Value	Meaning
9	512%	Pack is initialized "new files first"
11	2048%	Pack is initialized to maintain date of last write
12	4096%	Pack is initialized as a read-only pack
14	16384%	Pack is initialized as a private/system disk

All other values are reserved.

Read Account Attributes

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-25%), the read/write attributes code.
3	CHR$(-1%), the read account attributes subfunction code.
4	CHR$(N%), where N% is the attribute type code for the account to be accessed. The following values are the currently defined attribute type codes:

Value	Meaning
0%	Lookup by index
1%	Quotas
2%	Authorized privilege mask
3%	Password
4%	Date/time information
5%	Name entry
6%	Nondisk quotas

You may also use type codes in the customer-defined range (128-255).

5-6	PPN of the account to be accessed.
7-8	CHR$(I%)+CHR$(SWAP%(I%)), where I% is the index number of the account to read. Used only if byte 4 is 0; otherwise, 0. An index of 0 returns the account's accounting data.
9-22	Reserved; should be 0.
23-26+	The name and unit number of the disk device where the account resides.
27-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1-6	Not used.
7-20	Account attribute data. The first byte contains the attribute type code, as passed in byte 4. If byte 4 is 0, the first byte returns the type code of the attribute found. The remaining 13 bytes contain the actual attribute data. See the Discussion for a description of the 13 data bytes of attribute codes 1, 2, and 4.
21-30	Not used.

Privileges Required

None	Read attributes 1, 2, and 4-191 in your own account. That is, you can read all DIGITAL-defined attributes except password as well as the first 64 user-defined attributes (128-191).
GACNT or GREAD	Read all attributes in group accounts.
WACNT or WREAD	Read all attributes in all accounts.

Possible Errors

The following error messages are possible with the read, write, and delete account attributes subfunctions of this call.

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT The account you specified does not exist.	5
?NOT A VALID DEVICE The device you specified does not exist.	6
?PROTECTION VIOLATION You do not have sufficient privilege to perform the specified subfunction.	10
?DISK PACK IS NOT MOUNTED The disk you specified is not mounted.	21

?DEVICE NOT FILE STRUCTURED 30
The device on which the file is open must be a disk.

The following error message can only occur with the read account attributes subfunction:

?END OF FILE ON DEVICE 11
The attribute you specified was not found. If you specified a lookup by index, the index is greater than the number of attributes.

Discussion

This call searches for the specified attribute type and returns the data as 7 words, beginning in byte 7. The first byte is the type code; the remaining 13 bytes are the actual attribute data.

You can also specify a search by index number by passing a value of 0 in byte 4. This type of search enables programs like BACKUP to read all the account attributes without trying each of the 255 possible type codes. The program can issue successive calls, incrementing the index value by 1 each time.

The layouts of the data returned in bytes 7-20 for attribute type codes 1, 2, 4, and 6 are listed below. The data shown is considered internal information and is subject to change without notice.

Type 1 Quota information

Byte	Meaning
7	1, the attribute type code
8	Detached job quota
9-10	Logged-out quota (LSB)
11-12	Logged-in quota (LSB)
13	Logged-in quota (MSB)
14	Logged-out quota (MSB)
15	Reserved
16	Current usage (MSB)
17-18	Reserved
19-20	Current usage (LSB)

Type 2 Authorized privilege mask

Byte	Meaning
7	2, the attribute type code
8	Reserved
9-16	Authorized privilege mask
17-20	Reserved

Type 4 Date/time information

Byte	Meaning
7	4, the attribute type code
8	Keyboard of last login (-1 if last login was detached)
9-10	Date of last login, in RSTS/E internal format
11-12	Time of last login, in RSTS/E internal format in bottom 11 bits. Flags in high 5 bits.
13-14	Date of last password change
15-16	Time of last password change, in bottom 11 bits. Flags in high 5 bits.
17-18	Date of account creation
19-20	Expiration date (-1 if no expiration)

Flags in bytes 11-12 are:

2048%	No password is required to log in to this account
Others	Reserved

Flags in bytes 15-16 are:

2048%	Password cannot be looked up
4096%	No dialup logins allowed
8192%	No network logins allowed
16384%	No interactive logins allowed (spawn and batch only)
32767%+1%	Captive account

Type 6 Nondisk quotas

Byte	Meaning
7	6, the attribute type code
8	Total job quota
9-10	RIB quota
11-12	Message quota
13-20	Reserved

Write Account Attributes

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-25%), the read/write attributes code.
3	CHR$(-2%), the write account attributes subfunction code.
4	CHR$(N%), where N% is the attribute type code for the account to be accessed. Values for attribute type codes are:

Value	Meaning
0%	Accounting Data
1%	Quotas
2%	Authorized privilege mask
3%	Password
4%	Date/time information
5%	Name entry
6%	Nondisk quotas

You may also use type codes in the customer-defined range (128-255).

5-6	PPN of the account to be accessed.
7-20	The new account attribute data. The first byte contains the attribute type code. The remaining bytes contain the actual attribute data. See the Discussion for a description of the 13 data bytes of attribute codes 1, 2, and 4.
21-22	Reserved; should be 0.
23-26+	The name and unit number of the disk device where the account resides.
27-30	Reserved; should be 0.

Data Returned

None.

Write Account Attributes
F0=-25

Privileges Required

GACNT Write attributes for accounts in the group

WACNT Write attributes for all accounts

Possible Errors

In addition to the general error messages listed in the read account attributes subfunction, this call returns the following errors:

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE The attribute block does not exist yet, and it cannot be added because the directory is full.	4
?PROTECTION VIOLATION You do not have sufficient privilege to perform this subfunction, or the disk you specified is write-locked.	10

Discussion

This call searches for the attribute type code you specify in byte 4. If no match is found, it attempts to allocate a new directory entry to hold the new attribute. Next, it writes the data passed in bytes 7-20 into the attribute block. See the Discussion in the previous subfunction, "Read Account Attributes," for a description of the data passed in bytes 7-20.

This call writes the data exactly as passed, with two exceptions:

- Authorized privilege mask (attribute type 2) -- When writing the mask, the call ignores any attempt to turn on privilege bits if the caller does not have the corresponding privilege currently in effect. This applies only to attempts to change a bit from OFF to ON. Writing a bit as ON is allowed without checking if it was ON already.

- Date/time information (attribute type 4) -- The last login fields are always left alone. This ensures that any logins to an account leave a trace that cannot easily be altered.

Delete Account Attributes

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-25%), the read/write attributes code.
3	CHR$(-3%), the delete account attributes subfunction code.
4	CHR$(N%), where N% is the attribute type code for the account to be accessed. Values for attribute type codes are limited to those in the customer defined range (128 to 255).
5-6	PPN of the account to be accessed.
7-22	Reserved; should be 0.
23-26+	The name and unit number of the disk device where the account resides.
27-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

GACNT	Delete attributes for accounts in the group
WACNT	Delete attributes for all accounts

Possible Errors

In addition to the general error messages listed in the read account attributes subfunction, this call returns the following errors:

Meaning	ERR Value
?PROTECTION VIOLATION You do not have sufficient privilege to perform this subfunction; or the disk you specified is write-locked; or you attempted to delete attributes in the DIGITAL reserved attribute type range.	10

?END OF FILE ON DEVICE 11
The attribute you specified was not found.

Discussion

This subfunction deletes an attribute block for a specified account. It searches for the attribute type specified in byte 4. If found, the attribute block is deleted. This call applies only to attribute type codes in the customer defined range (128 to 255).

Add/Delete CCL Command

Data Passed

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

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-24%), the code to add/delete CCL.
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	PPN under which program to run is stored.
7-10	File name, in Radix-50 format, of the program to run.
11-12	File type, 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	Reserved; should be 0.

Add/Delete CCL Command
F0=-24

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

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-24%), the code to add/delete CCL.
3	CHR$(-2%) to delete a CCL command.
4	CHR$(U%), where U% is the number of unique characters in the command. U% must be between 1 and the length of the command and defines its abbreviation point.
5-12	Reserved; should be 0.
13-21	CCL command to delete.
22-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL

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

This call adds and deletes CCL commands. Chapter 10 of this manual describes the operation and design of CCL commands.

The command can be a string from one to nine characters long. The allowed single-character commands are A through Z, the at sign (@) character, the dollar sign ($) character, and the number sign (#) character. For commands longer than one character, the string must begin with a letter, and the remaining characters can be letters or digits. The command cannot begin with a numeric character because BASIC-PLUS interprets digits at the beginning of a line as a line number, not a command.

Commands have an abbreviation point after the first character. The abbreviation point is specified by the value in byte 4. If you specify an abbreviation point that equals the number of characters in the command, the command cannot be abbreviated. An example of an abbreviated CCL command is DIR (the abbreviation point follows the R), which uniquely defines the CCL command DIRECTORY. Any of the following abbreviations are also valid: DIR, DIRE, DIREC, DIRECT, DIRECTO, DIRECTOR, and DIRECTORY. If the abbreviation point for DIRECTORY follows the Y, then no abbreviation is valid.

Because of the way RSTS/E interprets CCL commands, you must make sure that you define similar commands in the correct order. For example, you must define MACRO before MAC. See the *RSTS/E System Manager's Guide* for more information about defining CCL commands.

Set Special Run Priority

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-22%), the code to set special run priority.
3-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

TUNE

Possible Errors

None.

Discussion

This SYS call 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 job keyboard monitor or whenever a program chains to another program. Thus, an appropriately privileged job can raise its priority without protecting against a user typing CTRL/C and retaining the higher priority.

Drop/Regain Temporary Privileges

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-21%), the code to drop temporary privileges.
3	If you do not specify a value, the call permanently drops temporary privileges. Otherwise, CHR$(N%), where N% means either of the following: 255% Temporarily drop temporary privileges. 0% Regain temporary privileges dropped by 255% value.
4-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

None.

Possible Errors

None.

Discussion

This call allows a program to selectively use temporary privileges. (See Chapter 1 for a description of temporary privileges.)

This call allows a program to activate temporary privileges for sections of code where they are needed, but take advantage of built in monitor protections (such as protection code arbitration) elsewhere. The call does not affect the permanent privileges of an account.

Good programming practice suggests two general approaches to using and controlling temporary privilege. If temporary privilege is required only for some initial set-up, the program can concentrate the code requiring privilege "up front" and then drop temporary privileges

permanently. The remainder of the program can then rely on the monitor's built-in protection, appropriate to the account the program is running in. The following sample code illustrates this approach:

```
10      V$ = SYS(CHR$(6%) + CHR$(-22%))
         !SET SPECIAL RUN PRIORITY - THIS REQUIRES PRIVILEGE

20      OPEN "$SYSTEM.FIL" FOR INPUT AS FILE 1%, MODE 8192%
         !OPEN A "REFERENCE" FILE, REGARDLESS OF PROTECTION
         !(USING READ-ONLY MODE, OFTEN GOOD PRACTICE, ALSO)

30      V$ = SYS(CHR$(6%) + CHR$(-21%))
         !HAVING DONE THE NECESSARY SET-UP, DROP TEMPORARY
         !PRIVILEGES FOR THE REMAINDER OF THE PROGRAM

40       .
         .
         .
```

A different approach is appropriate when a program needs temporary privileges at several points during execution. In this case, good programming practice suggests that temporary privileges be dropped early, and then regained just long enough to be used where needed. The following sample code illustrates this approach. (This sample uses line numbers appropriate for a program designed to be invoked by CCL. See Chapter 10 for more information on these conventions.)

```
1       EXTEND
2       PRINT '?PLEASE USE THE "xxxxx" CCL COMMAND'
\       GOTO 32767 !DISALLOW SOMEONE INVOKING THE PROGRAM BY RUN
          .
          .
          .
30000     !CONVENTIONAL CCL ENTRY POINT

        DROP.PRIVILEGES$ = CHR$(6%) + CHR$(-21%) + CHR$(255%)
         !COMPOSE THE "DROP PRIVILEGES" CALL STRING

\       V$ = SYS(DROP.PRIVILEGES$)
         !GO AHEAD AND DROP THEM, FIRST THING

\       REGAIN.PRIVILEGES$ = CHR$(6%) + CHR$(-21%) + CHR$(0%)
         !COMPOSE THE "REGAIN PRIVILEGES" CALL STRING,
         !FOR LATER USE
          .
          .    (FOLLOWING CODE CAN NOW EXECUTE
          .     WITHOUT PRIVILEGE)
          .
         !NOW, YOU REACH A POINT WHERE PRIVILEGE IS REQUIRED
         !(OPEN A PROTECTED FILE)
```

```
\       V$ = SYS(REGAIN.PRIVILEGES$) !GET PRIVILEGES TEMPORARILY
\       OPEN "$SYSTEM.FIL" FOR INPUT AS FILE 1%, MODE 8192%
         !OPEN A "REFERENCE" FILE, REGARDLESS OF PROTECTION
         !(USING READ-ONLY MODE, OFTEN GOOD PRACTICE, ALSO)
\       V$ = SYS(DROP.PRIVILEGES$) !AND DROP PRIVILEGES AGAIN
          .
          .  (AND SIMILARLY FOR OTHER OPERATIONS
          .   THROUGHOUT THE PROGRAM)
          .
32767   END
```

Lock/Unlock Job in Memory

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-20%), the lock/unlock a job in memory code.
3	CHR$(N%), where N% is 0% for lock and 255% for unlock.
4-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

TUNE

Possible Errors

None.

Discussion

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

You may want to use this call in a program with certain time-sensitive routines. The locked time must be very short to avoid degrading 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
```

Set Logins

Data Passed

Bytes	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	Reserved; should be 0.

Data Returned

Bytes	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.

Privileges Required

SWCTL

Possible Errors

None.

Discussion

This call sets the number of allowable logins to the number specified in byte 3. A value of 0 sets the number of allowed jobs to 1. The upper limit for the number of logins is either the system JOB MAX or the number of jobs that can currently be swapped, whichever is lower. If you specify a larger value, the system sets the number of logins to the upper limit. You do not receive an error.

The number of jobs that can log in to 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 that 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 some other keyboard.

Manipulate RTS, Resident Library, Dynamic Region
F0=-18 (UU.RTS)

Manipulate Run-Time System, Resident Library, Dynamic Region

This call has the following subfunctions:

- Add Run-Time System
- Remove Run-Time System
- Unload Run-Time System
- Add Resident Library
- Remove Resident Library
- Unload Resident Library
- Create Dynamic Region

Add a Run-Time System

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-18%), the run-time system manipulation code.
3	CHR$(N%), where N% is: 0% Use values for all bytes as specified in this call. 128% Use values defined in the .RTS file for bytes 13-14, 15-16, 19-20, and 21-22.
4	Reserved; should be 0.
5-6+	PPN of the file to add; if none is specified, [0,1] is the default.
7-10+	Run-time system name in Radix-50 format.
11-12	CHR$(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% and the run-time system requires a fixed address (read/write or /STAY run-time system), the monitor finds the address progressing from high to low memory. Otherwise, the monitor uses an area of memory calculated when the run-time system is actually needed.

If A% is -1%, the monitor calculates a fixed address, regardless of whether or not the run-time system requires one.

13-14 Maximum allowed user image size, in K words (the P.SIZE symbol). If byte 3 is 128%, these bytes are ignored.

15-16 Minimum allowed user image size, in K words (the P.MSIZ symbol). If byte 3 is 128%, these bytes are ignored.

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 call places the description block immediately after that of the primary RTS. If P% is a nonzero value less than or equal to the number of blocks currently in the list, the call places this new block in that position following the primary RTS block. If P% is 0% or a value greater than the number of blocks currently in the list, the call places this new block 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. If S% is 0%, the memory occupied by this RTS can be released as user job space whenever the usage count of the RTS goes to 0.

19-20 CHR$(F%) + CHR$(SWAP%(F%)), where F% is a flag word whose bits define this run-time system's characteristics. If byte 3 is 128%, these bytes are ignored. Only the high byte is used for flag bits. F% is the sum of the bits set as follows:

Value	Meaning
256%	This RTS is a keyboard monitor.
512%	This RTS handles only one user; that is, it is not shared by multiple users.
1024%	This RTS allows read and write access to its memory rather than read-only access.
2048%	Errors that occur under the control of this RTS should not 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 monitor computes the proper job image size (in K words) for any program running under this RTS as (file-size+3)/4.
	16384%	Reserved; should be 0.
	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 type, in Radix-50 format, for this run-time system (the P.DEXT symbol). If byte 3 is 128%, the call ignores these bytes.
23-24+		Name of the device (must be disk) on which the run-time system file is stored. If you do not specify a name, the call uses SY:.
25+		Unit number.
26+		Unit number flag.
27-30		Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL

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. See the discussion of assigning and allocating memory in the *RSTS/E System Installation and Update Guide* for guidelines on how to avoid fragmenting memory.	4

This error can also occur when the monitor attempts to determine the address assignment but cannot find any valid load address due to lack of memory.

?CAN'T FIND FILE OR ACCOUNT 5
A file with the name specified in bytes 7 through 10 and a file type of .RTS cannot be found in the account and device specified in this call (bytes 5-6 and bytes 23-26).

?PROTECTION VIOLATION 10
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.

?NAME OR ACCOUNT NOW EXISTS 16
A run-time system with the same name currently exists.

?ILLEGAL BYTE COUNT FOR I/O 31
The range of memory starting at the load address given in bytes 11 and 12 is not available. See the SYSTAT memory status report to select an available range of memory.

?NO BUFFER SPACE AVAILABLE 32
Adding a run-time system description block requires a small buffer and one is not currently available.

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 primary run-time system (RSX) and the default keyboard monitor (DCL) are transient from one time-sharing session to another. Thus, systems that offer auxiliary run-time systems must define them for each time-sharing session.

Remove a Run-Time System

Data Passed

Bytes	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	Reserved; should be 0.
7-10+	Run-time system name in Radix-50 format.
11-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL

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 primary RTS or the system default keyboard monitor and cannot be removed by this call.	10

Discussion

This call removes a run-time system from memory, deletes the monitor structure that defines this run-time system, and closes the run-time system file. The SHUTUP system program automatically performs these actions when it terminates time-sharing operations.

Unload a Run-Time System

Data Passed

Bytes	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	Reserved; should be 0.
7-10+	Run-time system name in Radix-50 format.
11-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL

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 by the job that is currently running. It cannot be unloaded now; a later attempt might succeed.	3
?CAN'T FIND FILE OR ACCOUNT The run-time system specified in bytes 7 through 10 is not currently defined.	5

Discussion

This call frees the portion of memory occupied by the run-time system. The memory is made available as user job space. The run-time system will be loaded again when it is needed. This function is valid for the primary run-time system, in which case it simply causes the run-time system to be re-read from disk. In all other cases, the unload function also clears the "stay" flag set when the run-time system was last added or loaded.

Add a Resident Library

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-18%), the resident library manipulation code.
3	CHR$(16%), add a resident library.
4	Reserved; should be 0.
5-6+	The PPN of the file to add; if none is specified, [0,1] is the default.
7-10+	The resident library name in Radix-50 format.
11-12	CHR$(A%)+CHR$(SWAP%(A%)), where A% is the 1K-word section of memory at which the resident library is to be loaded. The numbering begins at the first available 1K-word section and ends at n-1 (where n is the total number of of 1K-word sections of memory on the system). If A% is 0% and the library requires a fixed address (read/write or /STAY library), the monitor finds the address progressing from high to low memory. Otherwise, the monitor uses an area of memory calculated when the library is actually needed. See the Discussion for restrictions on specifying A%=0%. If A% is -1%, the monitor finds the first free space large enough to hold the resident library, starting from the top of memory.
13-17	Reserved; should be 0.
18	CHR$(S%), where S% is the stay flag. S% can be one of the following values:

Value	Meaning
0%	The memory occupied by this library can be freed for user job space whenever the usage count of the RTS is 0 (no active task is accessing the library).
128%	The library is made permanently resident.

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:

Value	Meaning
256%	Reserved; should be 0.
512%	The resident library is available to only one user. It is not shared by multiple users.
1024%	The resident library allows read/write access to its memory, rather than read- only access.
2048%	The resident library does not record errors in its code in the system error log.
4096%	The resident library is immediately removed from memory when its usage count equals zero.
8192%	Reserved; should be 0.
16384%	Reserved; should be 0.
32767%+1%	Reserved; should be 0.

21-22+ Protection code for the installed resident library. To specify a protection code, place a nonzero value in byte 21 and the protection code in byte 22. To accept the default protection, specify 0 in byte 21. The default protection code 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 Reserved; should be 0.

Data Returned

No meaningful data is returned.

Add a Resident Library
F0=-18

Privileges Required

INSTAL

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 so that memory would be fragmented and a swapping violation would occur. See the *RSTS/E System Installation and Update Guide* for guidelines on avoiding memory fragmentation. This error can also occur when the monitor attempts to determine the address assignment but cannot find any valid load address due to lack of memory.	4
?CAN'T FIND FILE OR ACCOUNT You specified a file name 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. Make sure that the file name you specify has a .LIB file type and is located in the specified account and device.	5
?PROTECTION VIOLATION The file you want 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 file name of a resident library that already exists.	16
?ILLEGAL BYTE COUNT FOR 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

This SYS call adds a specified library to the monitor's list of resident libraries. This call is similar to that used to add a run-time system.

If you specify a value of -1 in bytes 11-12, the monitor automatically decides where to load the resident library, finding the first free space large enough to hold the library, starting from the top of memory. The library file does not have to reside in account [0,1]; however, the file type must be .LIB.

If you specify a value of 0 in bytes 11-12, and you add the library neither read/write nor /STAY, the monitor calculates an address for the library when it is actually needed. This type of library has the following restrictions:

- Only 1 such resident library may be mapped by a program at any time
- A program mapping to the library must be running under the NULL run-time system.
- The maximum size of the library is 28K words.
- The start address for mapping the library may not be any higher than:

  32K - (size of library rounded up to the next highest 4K boundary)

See the *RSTS/E Task Builder Reference Manual* for more information on creating and using resident libraries.

Remove a Resident Library

Data Passed

Bytes	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	Reserved; should be 0.
7-10+	The resident library name in Radix-50 format.
11-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL

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

This SYS call removes a library from physical memory, deletes the monitor structure that defines the library, and closes the library file.

Unload a Resident Library

Data Passed

Bytes	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	Reserved; should be 0.
7-10+	The resident library name in Radix-50 format.
11-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL

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 an undefined resident library name in bytes 7 through 10.	5

Discussion

This SYS call removes a library from memory and frees that portion of memory for use by other jobs. The system reloads the library when it is needed. If the "stay" flag has been set by a previous add or load function, the call clears it.

Create Dynamic Region

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-18%), the run-time system manipulation code.
3	CHR$(24%), create dynamic region.
4-6	Reserved; should be 0.
7-10+	Region name in Radix-50 format. If zero is passed, this creates an unnamed dynamic region. See Discussion for information on unnamed dynamic regions.
11-12	CHR$(A%)+CHR$(SWAP%(A%)), where A% is the 1K-word section of memory at which this dynamic region is to be loaded. The numbering begins at 0 and ends at n-1 (where n is the total number of 1K-word sections of memory on the system). If A% is 0%, the monitor finds the first free space large enough to hold the region, starting from the top of memory.
13	Size of region in K-words, between 1 and 127 K. If you include a value of 128%, the monitor creates the region even if the full amount of memory requested is not available.
14-16	Reserved; should be 0.
17	CHR$(N%), where N% can be: 0% Do not attach job to region. 128% Attach job to region.
18	CHR$(N%), where N% can be: 0% Delete region when all users detach. 128% Do not delete region when all users detach.
19-20	CHR$(N%)+CHR$(SWAP%(N%)), where N% can be: 0% The region can be shared. 512% The region cannot be shared.
21	Protection code flag. If set, the protection code is real.

Bytes	Meaning
22	Protection code of region.
23-30	Reserved; should be zero.

Data Returned

Bytes	Meaning
5-6	Region ID.
13	Size of the created region, in K words.

Privileges Required

INSTAL

Possible Errors

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE If loaded at the address specified, memory would be fragmented and a swapping violation would occur. If the monitor is choosing the address, there is not enough free memory to create the region.	4
?NAME OR ACCOUNT NOW EXISTS You specified a name of a dynamic region or resident library that already exists.	16
?ILLEGAL BYTE COUNT FOR I/O You attempted to create a region of invalid size. Also, if the caller specified a load address, the load address was not valid.	31
?NO BUFFER SPACE AVAILABLE A small buffer was not available for the region description block. Also, if attachment was specified, a small buffer was not available for the window descriptor block.	32

Bytes	Meaning
22	Protection code of region.
23-30	Reserved; should be zero.

Data Returned

Bytes	Meaning
5-6	Region ID.
13	Size of the created region, in K words.

Privileges Required

INSTAL

Possible Errors

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE If loaded at the address specified, memory would be fragmented and a swapping violation would occur. If the monitor is choosing the address, there is not enough free memory to create the region.	4
?NAME OR ACCOUNT NOW EXISTS You specified a name of a dynamic region or resident library that already exists.	16
?ILLEGAL BYTE COUNT FOR I/O You attempted to create a region of invalid size. Also, if the caller specified a load address, the load address was not valid.	31
?NO BUFFER SPACE AVAILABLE A small buffer was not available for the region description block. Also, if attachment was specified, a small buffer was not available for the window descriptor block.	32

Discussion

A dynamic region is a portion of memory that is used to store data. This SYS call creates a dynamic region of memory. You can create two types of regions: named and unnamed.

A named dynamic region is typically used when multiple programs desire attachment to the region. The name of the region must be unique.

An unnamed dynamic region is used when a program wants exclusive use of an area of memory. When the monitor detaches from an unnamed dynamic region, it always removes the region from memory. In addition, the caller is automatically attached to a dynamic region on creation.

If you specify attachment to the region or create an unnamed region, the region ID is returned in bytes 5-6. This region is used in subsequent .PLAS directives. See the *RSTS/E System Directives Manual* for more information. Since BASIC-PLUS does not support the .PLAS directive, only named dynamic regions are useful to BASIC-PLUS programs.

Associate a Run-Time System with a File

Data Passed

Bytes	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	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

None	Specify a file with a protection code that permits write access
GWRITE	Specify a file in any account within the group
WWRITE	Specify any file
SYSIO	For files in [0,*] accounts

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

Associate a Run-Time System with a File
F0=-17

Discussion

This SYS call 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 larger than 65535 blocks (see Discussion in the section "File Utility Functions, SYS -26"), 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 by RSTS/E.

Shut Down System

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-16%), the system shut down code.
3-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

SHUTUP

Possible Errors

See Discussion.

Discussion

This SYS call logs out the current job (as does the FIP system function call code 5). In addition, this call bootstraps the initialization code after the job is logged out.

Before this SYS call can execute properly, several system conditions must be true:

- Only one job can be running on the system when you invoke the SYS call.
- The number of logins allowed on the system must be 1; that is, LOGINS DISABLED. (See Disable Further Logins, SYS -2).
- No disks except the system disk can be mounted.
- No files can be open on the system disk.

If all of these conditions are met, 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).

Accounting Dump

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-15%), the accounting dump code.
3-4	Reserved; should be 0.
5-6+	PPN of the account to which the system dumps the accumulated usage data. See Discussion. If both bytes are zero, the data is dumped to the current account.
7-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

GACNT	Access any account within the group
WACNT	Access any account

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 call allows a program to dump accumulated accounting data to the account specified in bytes 5 and 6. This enables user-callable utility programs to run on an account different from the account that called them and still charge the calling account for the time accumulated by the utility.

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, do the following:

1. Dump accounting data to the current account. This procedure zeros the data.

2. Perform processing for the account to be charged.

3. Dump accounting data to the account to be charged.

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

Change Date and Time

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-14%), the change date and time 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	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

DATES

Possible Errors

None.

Discussion

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

The execution of this function causes the monitor to awaken all sleeping jobs to inform them of the date/time change.

Note that you cannot specify a date earlier than 1-Mar-85.

Change Priority, Run Burst, and Maximum Size

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-13%), the change priority, run burst, and maximum size code.
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 nonzero 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 nonzero to indicate a change to the parameter as specified in byte 7. See Discussion.
7	CHR$(R%), where R% is the run burst. R% should be a value from 1% to 127%. When you specify a value outside this range, the monitor sets the run burst to 6.
8	CHR$(A%), where A% is 0% to indicate no change to the parameter in byte 9 or is nonzero 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 system uses the value of SWAP MAX. See Discussion.

Data Returned

No meaningful data is returned.

Privileges Required

TUNE

Possible Errors

Meaning	ERR Value
?ILLEGAL SYS() USAGE The specified job number does not exist.	18

Discussion

This call allows a user with TUNE privilege 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 called the job's run burst. It 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 to 2K words and can grow during processing to a size limited by the value of SWAP MAX. The system manager determines the size of SWAP MAX. (See the discussion of the START and DEFAULT options in the *RSTS/E System Installation and Update Guide*.) 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 64K words. A job can expand to 64K words with user I&D space when SWAP MAX is 64K words. Note that BASIC-PLUS jobs can never grow beyond 16K regardless of the job's maximum size. Therefore, the appropriately 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.

You must specify values for each of the variables in the parameter string. In the description of the data passed, the value A% before the related parameter variable determines whether that parameter changes or remains unchanged.

The system does not perform error checking on the data passed by the user. Values are used as passed even if they produce illogical results. For instance, if you specify a priority that is not a multiple of 8, its value is truncated to the next lowest multiple of 8. A priority greater than 127 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 causes the monitor to set the job's run burst to 6. Setting a compute-bound job's run burst to some high number tends to lock out other jobs. However, you do not

override the system maximum by setting S% to 255% or any value greater than SWAP MAX.

The monitor uses 256 queues to schedule jobs and polls each queue in sequential priority order. The monitor chooses the highest priority runnable job as the job to be run. Thus, a high priority compute-bound job can monopolize system resources.

The following rules apply for a job running on a pseudo keyboard:

- The job can never lower the priority of the controlling job below its own priority.
- The job can never raise its own priority above the priority of the controlling job. Any attempt to do so causes the system to set both priorities to the priority of the controlling job.

Get Monitor Tables - Part II

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-12%), the get monitor tables - part II code.
3-30	Reserved; should be 0.

Data Returned

Bytes	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	(DSKLOG) - The disk logical table.
23-24	(DEVSYN) - Start of synonym names in DEVNAM.
25-26	(MEMSIZ) - The word containing the size of memory physically present on the system. Size is in K words times 32.

27-28 (CCLLST) - The root link word in the linked list of concise command language (CCL) description blocks.

29-30 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.

Privileges Required

None.

Possible Errors

None.

Discussion

The three Get Monitor Table SYS calls return to your program either an address or a data value. The calls are commonly used with the PEEK function to read various system parameters and tables that 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. For a description of Get Monitor Tables - Part I, see SYS call -3. For a description of Get Monitor Tables - Part III, see SYS call -29. The section "The PEEK Function" describes the use of the PEEK function for certain convenient programming operations.

This call denotes each item of information described by a name in all uppercase letters. This name is the same one used to identify the information in the RSTS/E assembly listings. If the name is in parentheses, the information returned is an address of the data described. If the name is not in parentheses, the information returned is the actual data value. For example, the Get Monitor Tables - Part I call returns CNT.KB-1 in byte 3. The value returned is the number of terminal lines minus 1 configured on the system. However, bytes 11 and 12 return (JOBTBL), the address of the table of jobs. Use the PEEK function to inspect the address.

Note

All information returned by the call described in this section is internal to RSTS/E and is subject to change at any time.

Change File Statistics

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-11%), change file statistics code.
3	CHR$(N%), where N% is the internal channel on which the file is open and must be between 1 and 12, inclusive.
4-5	Date of last access to place in the file's accounting entry*. Specify the date as CHR$(D%)+CHR$(SWAP%(D%)), where D% is in the form required by the BASIC-PLUS DATE$(D%) function. (See the sample program in the Discussion.)
6-7	Date of creation to place in the file's accounting entry. Specify the date as CHR$(D%)+CHR$(SWAP%(D%)), where D% is in the form required by the BASIC-PLUS DATE$(D%) function. (See the sample program in the Discussion.)
8-9	Time of creation to place in the file's accounting entry. Specify the time as CHR$(T%)+CHR$(SWAP%(T%)), where T% is in the form required by the BASIC-PLUS TIME$(T%) function. (See the sample program in the Discussion.)
10-30	Reserved; should be 0.

*The DSKINT initialization option or the INITIALIZE command can change the meaning of date of last access to date of last modification for some disks. The SHOW DISK command tells which disks record the date of last modification.

Data Returned

No meaningful data is returned.

Privileges Required

DATES	Modify date of last access (other fields require no privilege)

Change File Statistics
F0=-11

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. Because the call does not supply default values, you must supply all three date and time values each time the call executes.

The following is a partial directory listing of an account, showing the file whose statistics are to be changed:

```
  CAT
 CTPBLD.BAS    0     60    30-Sep-84 30-Sep-84  03:13 PM

  Ready
```

The following program changes the date and time of creation to 12:00 noon, 21-Jul-85, and the date of last access to 21-Jul-85, as shown in the partial directory listing following the program:

```
  10       D% = 15202%
           !21-JUL-85 IS (202) + ((1985-1970)*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
2000     CLOSE #1%
32767    END

Ready

RUNNH

Ready

CAT
CTPBLD.BAS    0     60    21-Jul-85 21-Jul-85 12:00 PM

Ready
```

Note that you can use either this call or SYS call -26, File Utility Functions, to change data in a file's accounting entry. However, the two calls work differently when you open a file, write to it, change the date of last access in the file's accounting entry, and then close the file.

When you use this SYS call to change the date of last access before closing the file, the system does not update the file's accounting entry when it closes the file. After the file is closed, the file's accounting entry contains the date specified in the call, not the current date, as the date of last access. Use SYS call -26 if you want the date of last access to be changed to the current date when the file is closed.

Hang Up a Dataset
F0=-9 (UU.HNG)

Hang Up a Dataset

Data Passed

Bytes	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. See Discussion for values.
5-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

HWCTL

Possible Errors

None.

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:

Value	Meaning
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 127 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.

Get Open Channel Statistics

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-8%), the get open channel statistics code.
3	CHR$(N%), where N% is the channel number (between 0 and 12) of either the Device Data Block (DDB) for nondisk devices, the Window Control Block (WCB) for the file system, or the File Control Block (FCB).
4	CHR$(S%), where S% is 0% or 1%. The value of S% determines the information returned on the job. See Data Returned.
5-30	Reserved; should be 0.

Data Returned

If S% is 0%:

Bytes	Meaning
1	The current job number times 2.
2	Not used.
3-4	Word 1 of either the DDB or WCB.
5-6	Word 2 of either the DDB or WCB.
.	.
.	.
.	.
.	.
27-28	Word 13 of either the DDB or WCB.
29-30	Word 14 of either the DDB or WCB.

If S% is 1%:

Bytes	Meaning
1	The current job number times 2.
2	Not used.
3	The number of users who have a file open in a mode other than read regardless.
4	The number of users who have a file open in read regardless mode.
5	The status byte, which contains the following internal flag information:

Bit Value	Meaning
1	Reserved.
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	File is a UFD.
128	File is marked for deletion.

6	The most significant bits (MSB) of the file size. If a nonzero number is returned, it indicates a file whose size is greater than 65535 blocks.
7-8	The least significant bits (LSB) of the file size (in blocks).
9-30	Not used.

Privileges Required

None.

Possible Errors

Meaning	ERR Value
?NOT A VALID DEVICE You requested FCB information (byte 4 is 1) for a nondisk file.	6
?I/O CHANNEL NOT OPEN No file or device is open on the channel specified.	9
?ILLEGAL SYS( ) USAGE You used a subcode other than 0 or 1 in byte 4.	18
?ILLEGAL I/O CHANNEL The channel specified is outside the range 0 to 12.	46

Discussion

This call returns information kept in the DDB, WCB or FCB. Note that these data structures are internal to RSTS/E and subject to change at any time.

Specifying 0 in byte 4 returns information kept in the DDB or WCB data structures.

Specifying 1 in byte 4 returns information kept in the FCB including open counts, status byte, and current file size. RMS uses this call to determine file characteristics.

For an alternative to this call, see the description of the STATUS variable in the *BASIC-PLUS Language Manual*.

Enable CTRL/C Trap

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-7%), the enable CTRL/C trapping code.
3-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

None.

Possible Errors

None.

Discussion

After a program executes this SYS call, the run-time system treats the first CTRL/C typed on any terminal belonging to the job as a trappable error (?Programmable ^C trap, ERR=28). Upon execution of the trap, the system passes control immediately to the numbered program statement that has been designated as the error-handling routine by the last execution of an ON ERROR GOTO statement. After the trap, the system disables CTRL/C trapping. To keep CTRL/C trapping in effect, you must execute the SYS call again.

However, such trapping of CTRL/C guarantees only that a defined set of statements is executed when you type a CTRL/C. 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 the error ?Programmable ^C trap (ERR=28) occurs. The variable ERL refers to the last error trapped by the program.

Enable CTRL/C Trap
F0=-7

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 you type CTRL/C 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.

Several methods are available to protect a program from CTRL/C aborts. For example, you can:

- Open the console terminal in binary input mode, MODE 1% (see Chapter 4).
- Detach the program.
- Open the console terminal with MODE 16% (see Chapter 4).

If one of these three actions occurs, program execution under the job is immune to any CTRL/C.

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 you type a CTRL/C. 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 ten-second sleep allows you 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 a Job SYS call (SYS 8), or the REMOVE/JOB command. In this example, you can stop the program after the original trap by typing CTRL/C within ten seconds. DIGITAL recommends that you design programs that trap CTRL/C characters 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. When the CTRL/C error condition is processed by BASIC-PLUS, it reenables output just prior to printing the Ready prompt. 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.

Poke Memory

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-6%), the poke memory code.
3-4	CHR$(A%)+CHR$(SWAP%(A%)), where A% is the address to change. The address must be an even number.
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	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

SYSMOD

Possible Errors

Meaning	ERR Value
?PROTECTION VIOLATION The job executing the call does not have SYSMOD privilege, 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. This is a dangerous capability, and is therefore heavily protected. It requires the SYSMOD privilege.

The poke call allows only full word changes. If you want to change a byte, read the word using the PEEK function (see the section "The PEEK Function" at the end of this chapter), change the desired byte, and rewrite the entire word (using the Poke Memory call).

Broadcast to a Terminal

Data Passed

Bytes	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.

Privileges Required

SEND

Possible Errors

Meaning	ERR Value
?PROTECTION VIOLATION The job does not have sufficient privilege, 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 call prints the data broadcast 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, a hung-up modem line, or a terminal for which broadcast is disabled (that is, SET TERMINAL/NOBROADCAST), the call returns control to the program.

The call takes no action, and does not generate a system error. In this case, RECOUNT is equal to the length of the string that was passed.

Because the actual number of bytes broadcast depends on the availability of small buffer space, the destination keyboard may 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. See the *BASIC-PLUS Language Manual* for information on RECOUNT.

The following sample program segment shows how you can use this SYS call and the RECOUNT variable to ensure transmission of complete messages:

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

While the RECOUNT variable is nonzero, the remainder of the string M$ is rebroadcast. When the complete message is broadcast and RECOUNT is 0, the program exits from the loop. The program also exits if RECOUNT is equal to the length of the string, which occurs if the terminal has broadcast disabled or is a disabled terminal.

Force Input to a Terminal

Data Passed

Bytes	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.

Privileges Required

SYSIO

Possible Errors

Meaning	ERR Value
?PROTECTION VIOLATION The job does not have sufficient privilege, 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 modem line, control is returned to the program. The system takes no action and does not generate a system error.

Because the actual number of bytes forced depends on the availability of small buffer space, the destination keyboard may not receive all of the bytes forced. Unlike a broadcast to the terminal, however, the system discards characters it cannot store. Thus, the program cannot determine how many characters were actually forced.

Get Monitor Tables - Part I

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-3%), the get monitor tables - part I code.
3-30	Reserved; should be 0.

Data Returned

Bytes	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 data blocks (DDB).
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) for disks. (UNTOWN) - The table of unit owners (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 bits) count of free blocks. Each word applies to a separate disk unit.

27-28 Current date in internal format.

29-30 (UNTOPT) - The table of unit options.

Privileges Required

None.

Possible Errors

None.

Discussion

The three Get Monitor Table SYS calls return either an address or a data value to your program. The calls are commonly used with the PEEK function to read various system parameters and tables that 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. For a description of Get Monitor Tables - Part II, see SYS call -12. For a description of Get Monitor Tables - Part III, see SYS call -29. The section "The PEEK Function" describes the use of the PEEK function for certain convenient programming operations.

In this section, a name in all uppercase letters denotes each item of information described. This name is the same one used to identify the information in the RSTS/E assembly listings. If the name is in parentheses, the information returned is an address of the data described. If the name is not in parentheses, the information returned is the actual data value. For example, the Get Monitor Tables - Part I call returns CNT.KB-1 in byte 3. The value returned is the number of terminal lines minus 1 configured on the system.

However, bytes 11 and 12 return (JOBTBL), the address of the table of jobs. Use the PEEK function to inspect the address.

Note

All information returned by the call described in this section is internal to RSTS/E and is subject to change at any time.

Disable Further Logins

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-2%), the disable further logins code.
3-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

SWCTL

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 in to 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 in to 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 some other terminal.

Enable Further Logins

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(-1%), the enable further logins code.
3-30	Reserved; should be 0.

Data Returned

Bytes	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.

Privileges Required

SWCTL

Possible Errors

None.

Discussion

This call sets the number of logins allowed to the maximum number possible, given that swap file space may have been added. The call returns this value in byte 3. The number of logins never exceeds that specified at start-up time (JOB MAX).

Create User Account

This call has two subfunctions:

- Create User Account (New Format)
- Create User Account (Old Format)

Create User Account (New Format)

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(0%), the create user account code.
3	CHR$(N%), where N% is the number of clusters to preextend the User File Directory (UFD). See the Discussion for the values you can use.
4	Flag byte. CHR$(128%) to specify disk quotas in the new format.
5-6	CHR$(N%)+CHR$(SWAP%(N%), where N% is the starting device cluster number for UFD. (Use -1 to place UFD at middle of disk).
7-8	PPN. The project number can be between 0 and 254 with the exception of account [0,1]; the programmer number can be between 0 and 254.
9-12	Password in Radix-50 format. For a long password, specify 0 in bytes 9-12. See the Discussion for the procedures to set a long password.
13-14	Logged-out quota (LSB).
15-16	Expiration date, in RSTS/E internal format: (day of year) + [(number of years since 1970) * 1000]. Specify 0 to indicate "no expiration."
17-18	Logged-in quota (LSB).
19	Logged-in quota (MSB).

20	Logged-out quota (MSB).
21-22	Reserved; should be 0.
23-24+	Device name.
25+	Unit number.
26+	Unit number flag.
27	CHR$(C%), where C% is user file directory (UFD) cluster size; 0 means use the pack cluster size. A negative value means use the absolute value of the number if it is larger than the pack cluster size. Otherwise, use the pack cluster size.
28	Reserved; should be 0.
29-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
17-18	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 0 of the UFD.
19-20	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 1 of the UFD.
21-22	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 2 of the UFD.
23-24	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 3 of the UFD.
25-26	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 4 of the UFD.
27-28	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 5 of the UFD.
29-30	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 6 of the UFD.

Privileges Required

GACNT Create an account within the group

WACNT Create any account

Possible Errors

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE The monitor cannot allocate one cluster for the UFD you are creating because the disk is too full.	4
?PROTECTION VIOLATION The PPN is [0,0], or either the project or programmer number is 255.	10
?FATAL SYSTEM I/O FAILURE The account has been entered and the directory has been preextended. However, the account has not been given a password or quota because an internal consistency check has failed. Submit an SPR along with a SAVRES of the disk if you get this error.	12
?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 nonzero 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
?ILLEGAL BYTE COUNT FOR I/O The number of clusters specified in byte 3 is less than 0 or greater than 7. Or, the position specified in bytes 5 and 6 is beyond the end of the disk or the UFD, if placed at the specified position, will extend beyond the end of the disk. See Table 1-8 for information on disk sizes.	31

?MISSING SPECIAL FEATURE 66
You issued the new format call on a disk with RDS1.1 or RDS0.0 disk structure (pre-V9.0). The call returns this error if you:

- Specify a logged-out quota of 0 or between 65536 and 16777214 (2^24-2)
- Specify a logged-in quota other than 16777215 (2^24-1)

Discussion

You can use this call to perform the following operations:

- Create accounts on any disk that is mounted.
- Preextend and position the UFD which contains the directory entries for all of the files for the account you are creating. You may improve performance by preextending UFDs; however, this takes up additional disk space if the directory space is not used. In general, positioning the directory at the middle of the disk improves system performance.
- Set logged-in and logged-out disk quotas for the account you are creating. These quotas define the amount of disk space an account may use while logged-in and logged-out, as well as the maximum amount of disk space a logged-in account may use.

Byte 3 specifies the number of clusters to preextend the UFD. This byte can contain the following values:

Bit	Meaning
0	Preextend 1 cluster.
1-7	Preextend specified number of clusters.

Any other value returns the error ?Illegal byte count for I/O (ERR=31).

Bytes 5 and 6 specify where on the disk to place the UFD. Device clusters are numbered from 0 to the maximum shown in Table 1-8. Note that you receive the error ?Illegal byte count for I/O (ERR=31). if the device cluster number you specify plus the number of clusters to preextend exceeds the disk size.

Bytes 9-12 are used to specify the password. To establish a long password, specify 0 in these bytes. The program should then issue a

Set Password SYS call (SYS 8) to establish the desired password for the account.

Bytes 13, 14, and 20 specify the logged-out quota. Bytes 17, 18, and 19 specify the logged-in quota. On RDS1.2 disks (V9.0 format), quota values must be in the range 0 to 16777215 (2^24-1), with 0 meaning no allocation allowed and 16777215 meaning unlimited, On RDS1.1 or RDS0.0 disks (pre-V9.0 disks), logged-out quotas must be in the range 1 to 65535 or 16777215 (meaning unlimited), and logged-in quotas must be 16777215.

The data returned in bytes 17-30 gives you the device cluster number for each cluster of the UFD. Check the data returned to determine if there was enough space on the disk to completely preextend the UFD.

The monitor tries to extend the UFD contiguously, but it allocates non-contiguously if it must. If the monitor finds at least one cluster, no error is returned. You must check the data returned to see if the number of clusters you specified were allocated (or to determine if they were allocated contiguously).

Create User Account (Old Format)

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(0%), the create user account code.
3	CHR$(N%), where N% is the number of clusters to preextend the User File Directory (UFD). See the Discussion for the values you can use.
4	Flag byte. CHR$(0%) to indicate old format disk quotas.
5-6	CHR$(N%)+CHR$(SWAP%(N%), where N% is the starting device cluster number for UFD. (Use -1 to place UFD at middle of disk).
7-8	PPN. The project number can be between 0 and 254 with the exception of account [0,1]; the programmer number can be between 0 and 254.
9-12	Password in Radix-50 format. For a long password, specify 0 in bytes 9-12. See the Discussion for the procedures to set a long password.
13-14	Disk quota as an unsigned number. See the section "Unsigned Integer Numbers" for a description of unsigned numbers. Use 0 for an unlimited quota.
15-16	Expiration date, in RSTS/E internal format: (day of year) + [(number of years since 1970) * 1000]. Specify 0 to indicate "no expiration."
17-22	Reserved; should be 0.
23-24+	Device name.
25+	Unit number.
26+	Unit number flag.

27	CHR$(C%), where C% is user file directory (UFD) cluster size; 0 means use the pack cluster size. A negative value means use the absolute value of the number if it is smaller than the pack cluster size. Otherwise, use the pack cluster size.
28-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
17-18	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 0 of the UFD.
19-20	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 1 of the UFD.
21-22	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 2 of the UFD.
23-24	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 3 of the UFD.
25-26	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 4 of the UFD.
27-28	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 5 of the UFD.
29-30	CVT%$(SWAP%(X%)), where X% is the device cluster number for cluster 6 of the UFD.

Privileges Required

GACNT	Create an account within the group
WACNT	Create any account

Possible Errors

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE The monitor cannot allocate one cluster for the UFD you are creating because the disk is too full.	4

?PROTECTION VIOLATION 10
The PPN is [0,0], or either the project or programmer number is 255.

?FATAL SYSTEM I/O FAILURE 12
The account has been entered and the directory has been preextended. However, the account has not been given a password or quota because an internal consistency check has failed. Submit an SPR along with a SAVRES of the disk if you get this error.

?NAME OR ACCOUNT NOW EXISTS 16
The account specified in the call currently exists on the device specified.

?ILLEGAL CLUSTER SIZE 23
The cluster size specified in the call is either greater than 16 or is nonzero and less than the pack cluster size. See the *RSTS/E System Manager's Guide* for a discussion of valid cluster size values.

?DEVICE NOT FILE STRUCTURED 30
The device specified is not a disk or the disk is open in non-file-structured mode.

?ILLEGAL BYTE COUNT FOR I/O 31
The number of clusters specified in byte 3 is less than 0 or greater than 7.

Or, the position specified in bytes 5 and 6 is beyond the end of the disk or the UFD, if placed at the specified position, will extend beyond the end of the disk. See Table 1-8 for information on disk sizes.

Discussion

You can use this call to perform the following operations:

- Create accounts on any disk that is mounted.
- Preextend and position the UFD which contains the directory entries for all of the files for the account you are creating. You may improve performance by preextending UFDs; however, this takes up additional disk space if the directory space is not used. In general, positioning the directory at the middle of the disk improves system performance.

- Set a logged-out disk quota for the account you are creating.
- Specify an expiration date for the account you are creating.

Byte 3 specifies the number of clusters to preextend the UFD. This byte can contain the following values:

Bit	Meaning
0	Preextend 1 cluster.
1-7	Preextend specified number of clusters.

Any other value returns the error ?Illegal byte count for I/O (ERR=31).

Bytes 5 and 6 specify where on the disk to place the UFD. Device clusters are numbered from 0 to the maximum shown in Table 1-8. Note that you receive the error ?Illegal byte count for I/O if the device cluster number you specify plus the number of clusters to preextend exceeds the disk size.

Bytes 9-12 are used to specify the password. To establish a long password, specify 0 in these bytes. The caller should then issue a Set Password SYS call (SYS 8) to establish the desired password for the account.

Bytes 13-14 specify the disk quota. Quota values must be in the range 0 to 65535. If you issue the old format call on an RDS1.2 disk (V9.0 format), the call sets the following quotas:

- Sets the logged-out quota to the value specified in bytes 13-14. If you specify 0, it sets the logged-out quota to 16777215.
- Sets the logged-in quota to 16777215.

The data returned in bytes 17-30 gives you the device cluster number for each cluster of the UFD. Check the data returned to determine if there was enough space on the disk to completely preextend the UFD.

The monitor tries to extend the UFD contiguously. If the monitor finds at least one cluster, no error is returned. You must check the data returned to see if the number of clusters you specified were allocated (or to determine if they were allocated contiguously).

Delete User Account
F0=1 (UU.DLU)

Delete User Account

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(1%), the delete user account code
3-6	Reserved; should be 0.
7-8	PPN. This call generates an error if you specify account [0,0] or [0,1].
9-22	Reserved; should be 0.
23-24+	Device name; must be a disk.
25+	Unit number.
26+	Unit number flag.
27-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

GACNT	Delete an account within the group
WACNT	Delete any account

Possible Errors

Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE For an account being deleted from the public structure, a user is currently logged in to the system under the account.	3
?CAN'T FIND FILE OR ACCOUNT The specified account does not exist.	5

?DEVICE NOT AVAILABLE 8
The disk is mounted /NOSHARE by another user.

?PROTECTION VIOLATION 10
Account specified is either [0,0] or [0,1].

?NAME OR ACCOUNT NOW EXISTS 16
The account contains files (it has not been zeroed).

?DEVICE NOT FILE STRUCTURED 30
Device specified is not a disk or is a disk open in non-file-structured mode.

Discussion

This call deletes a user account from a private disk or the public structure. If the error ?Device not available (ERR=8) occurs, you must first delete all files in the account and release the UFD clusters with the Zero a Device SYS call (SYS 13) or the DCL DELETE command.

Disk Pack Status

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(3%), the disk pack status code.
3	CHR$(N%); the following values of N% determine the action performed:

Value	Action
0%	Mount a disk pack or cartridge.
2%	Dismount a disk pack or cartridge.
4%	Restrict a disk pack or cartridge.
6%	Unrestrict a disk pack or cartridge.
8%	Load SAT of a disk into memory.
10%	Unload SAT of a disk from memory.

For all values of N%:

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 that 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	First six characters of the logical name for this disk, in Radix-50 format (see bytes 19-20). If bytes 11-12 are 255% the logical name given here and in bytes 19-20 replaces 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:

Value	Meaning
256%	Mount disk /NOQUOTA.
1024%	Mount with pack identification lookup. See Discussion.
2048%	Mount disk for one user only (/NOSHARE).
4096%	Mount disk read/write even if disk was initialized as read-only.
8192%	Mount disk for read-only access.
16384%	Mount as a private disk (/PRIVATE).

Note that you can combine the various mode bits where appropriate.

19-20 Last three characters of the logical name for this disk, in Radix-50 format (see bytes 13-16). When you use a logical name of fewer than 9 characters, you must fill the extra space with blanks. If bytes 11-12 are 255%, the logical name given here and in bytes 13-16 replaces 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.

Data Returned (Mount a disk pack or cartridge)

Bytes	Meaning
1	The current job number times 2.
2-12	Not used.
13-16	First two words of the logical name used for this disk, in Radix-50 format.
17-18	Not used.
19-20	Third word of the logical name used for this disk, in Radix-50 format.
21-30	Not used.

Disk Pack Status
F0=3

Data Returned (Load SAT into Memory)

Bytes	Meaning
1	The current job number times 2.
2-12	Not used.
13-14	The amount of memory used (in bytes).
17-30	Not used.

Privileges Required

HWCFG	Declare a mounted disk as restricted or unrestricted
MOUNT	Mount or dismount a disk /SHARE; Dismount a disk owned (/NOSHARE) by another account; Mount a disk /NOSHARE for a job running under another PPN; Mount a dirty disk
SWCTL	Load SAT of a disk into memory or unload SAT of a disk from memory

Possible Errors

Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE An attempt is made to dismount a disk that 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
?I/O CHANNEL ALREADY OPEN An attempt was made to load the SAT of a disk into memory more than once.	7
?I/O CHANNEL NOT OPEN You tried to unload a SAT that was not loaded.	9
?PROTECTION VIOLATION An attempt was made to mount a disk that does not contain the RSTS/E file structure. Use either the INITIALIZE command, the online DSKINT program, or the DSKINT initialization option to initialize the	10

disk. Or, you do not have the required privilege for the attempted operation.

?DEVICE HUNG OR WRITE LOCKED 14
An attempt was made to mount a disk read/write that is not write enabled. Or, an attempt was made to load the SAT of a read-only disk.

?ILLEGAL SYS() USAGE 18
An attempt to mount a disk that is already mounted or that resides in a non-dismounted drive; or disk specified is the system disk.

?PACK IDS DON'T MATCH 20
An attempt is made to mount a disk with an incorrect pack label.

?DISK PACK IS NOT MOUNTED 21
An attempt is made to lock, unlock, or dismount a disk that is not mounted. Or, an attempt is made to load or unload the SAT of a disk that is not mounted.

?DISK PACK NEEDS REBUILDING 25
The storage allocation table on the disk needs to be restructured because the disk was not properly dismounted when it was last used. Before using the disk, use the MOUNT command or the ONLCLN program to rebuild the storage allocation table. Note that when this error occurs, the disk is always mounted read-only with the "dirty" bit set.

?FATAL DISK PACK MOUNT ERROR 26
The disk structure is invalid. For example, the cluster size is larger than 16 or the storage allocation table is unreadable.

?DEVICE NOT FILE STRUCTURED 30
An attempt is made to restrict, unrestrict, or dismount a disk currently opened in non-file-structured mode. Or, an attempt is made to load or unload the SAT of a disk currently opened in non-file-structured mode.

?NO BUFFER SPACE AVAILABLE 32
An attempt was made to load the SAT of a disk into memory and no memory is available.

Discussion

This call lets you mount, dismount, restrict, and unrestrict a disk, load the Storage Allocation Table (SAT) of a disk into memory, or unload the SAT of a disk from memory. For a discussion of disk management on RSTS/E, see the *RSTS/E System Manager's Guide*.

The load SAT subfunction is used to load the SAT of a disk into memory. Loading the SAT of a unit reduces the amount of disk I/O which the file processor needs to do, thereby increasing overall system throughput. The unload SAT subfunction frees XBUF for other use.

The mode value in bytes 17 and 18 of the mount call modifies the mount operation:

- MODE values 16384% and 8192% correspond to the /PRIVATE and /NOWRITE qualifiers of the DCL MOUNT command.
- MODE value 4096% mounts packs initialized as /NOWRITE in read/write mode (normally such packs are mounted read-only). This mode is used by the /WRITE qualifier of the DCL MOUNT command.
- MODE value 2048% mounts the disk for a single user. Only that user can access the disk; other users receive the error ?ACCOUNT OR DEVICE IN USE (ERR = 3). In addition, mounting a disk /NOSHARE disables any privileged programs on that disk. The system automatically dismounts the disk when the user logs out or the job is killed.
- MODE value 1024% mounts a disk without specifying the pack identification label. The system looks up the pack identification label on the disk.
- MODE value 256% mounts a disk /NOQUOTA. This instructs the monitor not to perform quota checking on the unit.

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.

Null 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 null bytes in the table. To notify the program that a logical name was not placed in the table, null characters are written in bytes 13-16 and 19-20 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 13-16 and 19-20. If bytes 13-16 and 19-20 contain null bytes, no name is placed in the table. When bytes 13-16 and 19-20 contain a logical name, the call performs the same actions as previously described 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.

Login/Verify Password

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(4%), the LOGIN code.
3	Reserved; should be 0.
4	CHR$(L%+P%), where:

L% indicates whether to perform a LOGIN or check a password. L% can be one of the following values:

Value	Meaning
0%	Perform the login function.
1%	Check the password only.
4%	Check the system password. See Discussion.
8%	Perform the login function without checking the password.

P% indicates the password format. P% can be one of the following values:

Value	Meaning
0%	The password is specified as 2 words of Radix-50 data (old format).
2%	The password is specified as 14 bytes of ASCII data (new format).

5-6+	PPN; must not be group [0,*].
7-20	Password of the account specified in bytes 5 and 6. If P% in byte 4 is 0, specify the password in bytes 7-10 in Radix-50 format, and pad the extra bytes with nulls. If P% in byte 4 is 2, specify the password as 14 bytes of ASCII data.
21-22	Reserved; should be 0.

23-24+	Device name; must be a disk. You must specify the device if you are verifying a user password (L%=1). This field is reserved for login (L%=0% or L%=8%) and verify system password (L%=4%). However, you do not need to specify the device if you are verifying a system password (L%=4).
25+	Device unit number.
26+	Unit number real flag.
27-30	Reserved; should be 0.

Data Returned for Login Function

Bytes	Meaning
1	The current job number times 2.
2	Flag byte. See Discussion.
3	Total number of jobs logged in to 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.

Data Returned for Check Password Function

No meaningful data is returned.

Privileges Required

None	Login to an account with the correct password
GACNT	For any account within the group: check password, or log in without checking password
WACNT	For any account: check password, or log in without checking password
DEVICE	Check any password on a restricted disk (required in addition to the GACNT/WACNT privilege)

Possible Errors

Meaning	ERR Value
?BAD DIRECTORY FOR DEVICE The account does not have all the necessary directory structures.	1
?ILLEGAL FILE NAME When verifying a password other than the system password, the given password does not match the account password.	2
?CAN'T FIND FILE OR ACCOUNT One of the following conditions occurred: o The PPN specified in the call is [0,1] or does not exist. o The password specified in the call does not match the password of the account on the system. o The system password specified in the call does not match the password block that exists in account [0,1].	5
?PROTECTION VIOLATION You tried to verify a user password but did not specify the device name and unit number.	10
?NO BUFFER SPACE AVAILABLE No buffers are available to create the necessary internal structures.	32

Discussion

This call performs three functions:

- Logs in a job. If the calling job is already logged in to 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. The caller specifies passwords either as two words of Radix-50 data (old format), or as 14 bytes of ASCII data (V9.0 format).
- Verifies a password while logged in. This function allows a program to verify a user-supplied password without having to log out and back in.
- Checks a system password.

Bit 2 of byte 4 is the system password flag. If bit 2 is set, then the system performs the check system password function. Successful completion of this function sets a "system password verified" flag in the job data structure, which is checked by the Login function. The caller passes the system password to check in bytes 7-20. The system follows these procedures when performing a system password check:

1. If the caller already has the "system password verified" flag set, then this function completes immediately.

2. The system checks whether the system password applies to this job. The system manager uses the SET SYSTEM/PASSWORD_PROMPT command to specify which jobs require the system password. For example, the system password may be set to apply only to dial-up and network jobs. In that case, local jobs do not require a system password. The function completes successfully, but it does not set the "system password verified" flag.

3. If a system password applies to the job, the system looks for the password block of the [0,1] account on the system disk. If there is none, the function completes successfully.

4. If a password block exists in account [0,1], then the system compares it against the specified value. If they match, the function completes successfully. Otherwise, it returns the error ?Can't find file or account (ERR=5).

Byte 2 is the flag byte. On the login function, this byte reports cases where the password and PPN are valid but the job or detached quota for the account are exceeded. Values are:

Value	Meaning
-1	Login succeeded
-2	Login rejected, detached job quota exceeded
0 or Positive	Login rejected, job quota exceeded

If the call returns a value other than -1, the job is still logged out.

Logout

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(5%), the LOGOUT code.
3-4	CVT%$(SWAP%(N%+N0%)), where N% and N0% can have the following values: N%=0% Close files, deassign devices, remove receivers, and dismount disks mounted /NOSHARE. N%=1% Log out without closing files, deassigning devices, removing receivers, and dismounting disks mounted /NOSHARE. N0%=0% Check detached job and disk quotas on all mounted disks before logout. N0%=2% Perform logout without checking quotas. N% is forced to zero if you do not have WACNT privilege. N0% is forced to zero if you do not have EXQTA privilege.
5-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1-2	Not used.
3-4	Logout status. The following values can be returned: 0 = No quota is exceeded. If you have WACNT privilege, the system returns control to your program, and you can examine the data returned. If you do not have WACNT privilege, the system does not return control to your program. Instead, the system kills your job after performing necessary clean-up functions.

-1 = Either the disk or detached job quota is exceeded; your job is still logged in. (Byte 13 tells you which quota is exceeded.)

-2 = A disk quota is exceeded; your job is logged out. If you have WACNT privilege, the system returns control to your program. If you do not have WACNT privilege, the system kills your job after performing necessary clean-up functions.

See the Discussion for an explanation of quota checking.

5-12 Not used.

13 If bytes 3-4 contain -1, or -2, this byte indicates which quota is exceeded by returning one of the following values:

0 = Disk quota.

1 = Detached job quota.

If a quota is exceeded, the following data is returned:

15 If byte 13 is 0, this byte returns the current disk quota (MSB) in blocks as an unsigned integer.

16 If byte 13 is 0, this byte returns the current disk usage (MSB) in blocks as an unsigned integer.

17 If byte 13 is 1, this byte contains the number of detached jobs currently active in the account.

18 If byte 13 is 1, this byte contains the number of detached jobs allowed on logout. (Byte 17 minus byte 18 is the number of detached jobs over quota.)

19-20 If byte 13 is 0, these bytes return the current disk quota (LSB) in blocks as an unsigned integer (see the section "Unsigned Integer Numbers").

21-22 If byte 13 is 0, these bytes return current disk usage in blocks (LSB) as an unsigned integer (see the section "Unsigned Integer Numbers").

23-24+ If byte 13 is 0, these bytes return the disk name as 2 ASCII characters as an unsigned integer (see the section "Unsigned Integer Numbers").

25+ If byte 13 is 0, this byte contains the unit number as 2 ASCII characters.

26+	Unit number flag.
27-30	Not used.

Privileges Required

None	Log out normally
WACNT	Log out without self kill
EXQTA	Suppress quota checks on logout (N0%=2% in bytes 3-4)

Possible Errors

None.

Discussion

The LOGOUT and LOGIN system programs use this call. It can close all open channels, deassign all devices, and clear the job from the monitor message table (depending on the values passed in bytes 3-4). In addition, the call updates statistics on the disk and disassociates the PPN from the job number. The call also enforces quotas on all disks at log-out time (all disks mounted read/write with quota checking enabled must be under quota at log-out time).

Note that if the caller has WACNT privilege, this call does not immediately terminate the job. Instead, the monitor terminates a logged-out job when the program the job is running finishes executing. If the caller does not have WACNT privilege, this call terminates the job immediately, effectively performing a self-kill.

If a quota is exceeded, byte 13 of the returned data indicates which quota (the system only reports this information for the first encountered quota exceeded). If the detached job quota is exceeded, the call does not perform the logout. If a disk quota is exceeded and bytes 3-4 are returned as -2, at least one other attached job is logged in to the account, and the call performs the logout with a quota warning. However, if a disk quota is exceeded and bytes 3-4 are equal to -1, the call does not perform the logout. Note that the calling program should examine bytes 3-4 to determine if the logout function was performed. For callers without sufficient privilege, control returns to the program only if the call did not perform the logout (bytes 3-4 are returned as -1).

Attach

This call has three subfunctions:

- Attach
- Reattach
- Swap Console

To use this call, you need to understand the concept of terminal ownership. A terminal is "owned" when:

- It becomes attached to a job by logging in. When data is entered at a free terminal, the system starts a job to handle the input and gives the job the next available job number. The system then starts the LOGIN program to allow the user to log in to the system. (See the *RSTS/E System User's Guide* for the operational details.)

  When a user is logged in to the system, the system associates the activated job with both the terminal at which the user is typing and the account number used for system identification. The job is then considered active on the system and in attached mode (or attached to the terminal). The system associates I/O channel 0 with the terminal that activated the job. The terminal associated with channel 0 is called the job's console terminal or console keyboard. A job can have only one console terminal, the keyboard to which it is attached.

- It is opened on a nonzero channel. A job can own several terminals that are open on nonzero channels.

- It is allocated for the use of a job (with the the DCL ALLOCATE command).

Attach

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(6%), the attach and reattach code. The attach code is the same as the reattach code, except that the format of the data passed is different. See the next section for the format of the Reattach SYS call.
3	The number of the job to attach to the terminal.
4	CHR$(N%+P%), where N% is 0 and P% can be one of the following values: P%=0% The password is specified as 2 words of Radix-50 data (old format). P%=2% The password is specified as 14 bytes of ASCII data (new format). P%=4% Suppress the password check.
5-6+	PPN of the job to attach to the terminal, or zero to specify the same PPN as the caller.
7-20	Password of the account specified in bytes 5 and 6 (not necessary if the PPN of the job being attached matches that of the caller). If P% in byte 4 is 0, specify the password in bytes 7-10 in Radix-50 format, and pad the extra bytes with nulls. If P% in byte 4 is 2, specify the password as 14 bytes of ASCII data and pad the the extra bytes with nulls. See Discussion.
21-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

None	Attach to a job running under the caller's PPN
GACNT	Attach to a job in another account within the same group
WACNT	Attach to a job in any account

Possible Errors

Meaning	ERR Value
?ILLEGAL SYS() USAGE One of the following conditions generates the error: o The job executing the call has an open channel. o The job executing the call is a source (.BAS) program rather than a compiled (.BAC) program. o The job number specified in byte 3 is not a detached job. o The account in the call does not match the PPN of the job being attached. o The job being attached has a PPN different from that of the caller, and the password does not match. o The job executing the call is detached. o The caller does not have sufficient privilege to attach to a job that is running under a different PPN than the caller.	18

Discussion

The LOGIN system program executes this call. See the description of the ATTACH command in the *RSTS/E System User's Guide* for an example of the call's use. Note that, if byte 3 is the number of the job executing the call, the system performs the reattach action. See the next section for a description of the reattach process.

If the job being attached has the same PPN as the caller, no password is needed. However, if the job being attached has a PPN different from the caller, the password is required unless the suppress password check flag is set (P%=4%).

Reattach

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(6%), the attach and reattach code. The reattach code is the same as the attach code, but the format of the data passed is different. See the previous section 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	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

DEVICE Reattach to a terminal that is a restricted device

Possible Errors

Meaning	ERR Value
?ILLEGAL SYS() USAGE One of the following conditions generates the error: o The job number specified in byte 3 is less than 1 or greater than the JOB MAX value on the system. o The job executing the call is not detached. o The keyboard number in byte 4 is out of range.	18

- The terminal specified by the keyboard number in byte 4 is currently assigned, opened, or the console keyboard of some job other than the calling job.

- The terminal specified is a restricted device and you do not have the DEVICE privilege.

Discussion

This call performs differently for users with or without DEVICE privilege. If a job with DEVICE privilege executes the reattach call, the call establishes the terminal specified in byte 4 as the job's console keyboard. In this manner, a job can reattach to a terminal after having detached.

This call is also available to users who do not have DEVICE privilege, with certain restrictions. If the job issuing the reattach request has the specified terminal assigned, the request is accepted. If the terminal is free (not assigned), the reattach is allowed only if the terminal has not been marked restricted (by the SET DEVICE command). If the terminal is marked restricted and is free, the job must have DEVICE privilege to reattach to it.

Swap Console

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(6%), the attach, reattach, and swap console code. The swap console code is the same as the attach and reattach code, but the format of the data passed is different. See the previous two sections for the attach and reattach formats.
3	CHR$(J%), where J% is the job number to swap with. If the calling job is attached, this job must be detached. If the calling job is detached, this job must be attached. Both the calling job and this job must be running under the same PPN.
4	CHR$(S%), where S% is 1 to indicate the swap console function.

Data Returned

No meaningful data is returned.

Privileges Required

None.

Possible Errors

Meaning	ERR Value
?ILLEGAL SYS () USAGE	18

One of the following conditions generates this error:

- Both the calling job and the job specified in byte 3 are detached, or neither job is detached.
- The job specified in byte 3 has a PPN different from the caller's.
- The value in byte 4 is neither 0 nor 1.

Discussion

The swap console call allows two jobs, one of which is detached, to exchange ownership of a console terminal. In effect, this call combines a detach of the attached job with a reattach of the detached job. Its purpose is to allow detached programs to temporarily obtain ownership of a console to perform certain functions at the request of the program running at that terminal. Once these functions are performed, ownership of the console can then be returned to the requesting program.

This call requires no privileges and can be executed by either a detached job or an attached job. Both jobs must be running under the same PPN. If the caller is detached, the job specified must be attached. If the caller is attached, the job specified must be detached. If these conditions are met, the job that was attached will be detached from its console, and the job that was detached will be attached to that console.

This call does not affect open files or ownership of devices with one exception: any channels open on KB:. These channels are affected because the job that was detached now has its console back, and the job that was attached no longer has a console. The effect is the same as an ordinary detach (with CLOSE option specified, see SYS call 7, Detach) or reattach operation.

Detach

Data Passed

Bytes	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:

J%=0%	Detach the calling job (also the case if byte 3 is not specified).
J%=1%-63%	Detach another job.

C% is the CLOSE flag:

C%=0%	Do not deassign the console or close its I/O channels (also the case if byte 3 is not specified).
C%=128%	Deassign the console and close all I/O channels on which the console is open after detaching the job.

Data Returned

No meaningful data is returned.

Privileges Required

None	Detach your own job
JOBCTL	Detach another job
EXQTA	Detach the job even if the detached job quota is exceeded

Possible Errors

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE There are no more job slots available, or no small buffers to create the new job.	4
?ILLEGAL SYS( ) USAGE The job is already detached.	18
?NO BUFFER SPACE AVAILABLE No small buffers are available for a context buffer to create the new job.	32
?QUOTA EXCEEDED You exceeded the detached job quota.	69

Discussion

To use this call, you need to understand the concept of terminal ownership. See the introduction to the Attach SYS call (SYS 6) for more information on terminal ownership.

This call disassociates the calling job or another job from its console keyboard. The following sample program segment prints a message and detaches itself from the keyboard:

```
100     PRINT "DETACHING..."
        ! NOTIFY THE USER
110     A$ = SYS(CHR$(6%) + CHR$(7%))
        ! DO THE DETACH
```

It is possible for a job to be detached while still maintaining ownership of a terminal.

By executing this call, a job can detach itself from its console terminal, or, if it has the required privilege, detach another attached job from its console terminal. After a job is placed in the detached state, it runs like any other job on the system, but it does not have access to its console terminal (on channel 0). The detached state is advantageous for noninteractive jobs. By detaching, the job frees a terminal for other use and becomes immune from interruption by CTRL/C. (See Chapter 4 for a description of MODE 16%, which prevents CTRL/C interruption and hibernation.)

The values passed in byte 3 specify the job number to detach and determine whether I/O channels are closed. The following paragraphs explain how the "close flag" (bit 7, value 128%) works.

If the detached job has its console terminal open on some nonzero

channel, and 128% is not specified in byte 3 of the data passed, the job can perform I/O on the keyboard from which it is detached. However, it must use a nonzero I/O channel. In addition, a terminal that is previously assigned remains assigned. If the detached job tries to perform I/O on channel 0, the system places the job in the hibernate state. (A job in the hibernate state is suspended until some user attaches to it.) A detached job that performs I/O to the detached keyboard on a nonzero channel, however, retains control of the terminal (I/O is performed). Thus, the terminal is not free for other use.

Specifying 128% in byte 3 of the data passed forces the system to disassociate the terminal from any nonzero I/O channel being used. This value also forces deassignment of the console terminal if it was previously assigned. The disassociation that the detach call performs thus 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 to the terminal on the nonzero channel causes the system to place the job in the hibernate state (or if the terminal was opened with MODE 16%, returns the error ?I/O to detached keyboard).

If a job running under the control of a DCL command file detaches itself using this call, a new job is created at that terminal, logged in to the same account, and execution of the command file continues with the new job. The call creates a new job only if the terminal is closed (with the close flag) or is not open on any channel. See the *RSTS/E Guide to Writing Command Procedures* for more information on writing DCL command files.

Change Quota, Password, Expiration Date

This call has five subfunctions:

- Change Quota (New Format)/Expiration Date/Password (Old Format)
- Change Quota (Old Format)/Expiration Date/Password (Old Format)
- Set Password (New Format)
- Kill Job
- Disable Terminal

Change Quota (New Format)/Expiration Date/Password (Old Format)

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(8%), the change quota, password, kill job, and disable terminal code.
3	Detached job quota.
4	Flag byte. See the Discussion for the values you can use.
5-6	Reserved; should be 0.
7-8	PPN. Zero for both values means the current account.
9-12	New password in Radix-50 format (pre-V9.0 format). All zeros means no change. See the next section, "Set Password," for a description of how to set passwords using V9.0 format.
13-14	Logged-out quota (LSB).
15-16	Expiration date, in RSTS/E internal format: (day of year) + [(number of years since 1970) * 1000]. CHR$(0%) indicates no change.
17-18	Logged-in quota (LSB).

19	Logged-in quota (MSB).
20	Logged-out quota (MSB).
21	CHR$(255%) to change any quota (see byte 4). CHR$(0%) to indicate that no change is to be made to the quota.
22	Reserved; should be 0.
23-24+	Device name. If no device name is specified, SY: is used.
25+	Unit number.
26+	Unit number flag.
27-28	Must be CHR$(0%).
29-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

GACNT	Change quota or expiration date within the group
WACNT	Change any quota or expiration date

Possible Errors

Meaning	ERR Value
?BAD DIRECTORY FOR DEVICE The account does not have all the necessary directory structures.	1
?ILLEGAL FILE NAME You specified a password that was less than six characters. Or, you specified a password that contained illegal characters (such as ?).	2
?CAN'T FIND FILE OR ACCOUNT The account is not present on the disk specified.	5

?NOT A VALID DEVICE 6
The device specification supplied in bytes 23 through 26 is illegal because the unit or its type is not configured on the system.

?ILLEGAL SYS() USAGE 18
The device specified is not a disk.

?MISSING SPECIAL FEATURE 66
You issued the new format call on a disk with RDS1.1 or RDS0.0 disk structure (pre-version 9.0). The call returns this error if you:

- Specify a logged-out quota of 0 or between 65536 and 16777214 (2^24-2)
- Specify a logged-in quota other than 16777215 (2^24-1)

Discussion

You can use this call to perform any or all of the following operations:

- Change logged-in and logged-out disk quotas for the account. These quotas define the amount of disk space an account can use while logged-in and logged-out, as well as the maximum amount of disk space a logged-in account can use.
- Change a user's password using the old format. To change a password using the new 14 character ASCII format, see the next section, "Set Password."

Byte 4 specifies the flag byte. This byte can contain the following values:

Value	Meaning
0%	Change disk quota using old format (see next subfunction)
1%	Change logged-out quota (new format)
2%	Change logged-in quota (new format)
4%	Reserved
8%	Change detached job quota (new format)
16%	Reserved
32%	Reserved
64%	Reserved
128%	Change disk quota using new format

Bytes 13, 14, and 20 specify the logged-out disk quota. Bytes 17, 18, and 19 specify the logged-in quotas. Quota values must be in the range 0 to 16777215 (2^24-1). On pre-version 9.0 disks, logged-out quotas must be in the range 1 to 65535 or 16777215, and the call rejects logged-in quotas other than 16777215.

Change Quota, Password, Expiration Date
F0=8

Change Quota (Old Format)/Expiration Date/Password (Old Format)

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(8%), the change quota, password, kill job, and disable terminal code.
3	Reserved; should be 0.
4	CHR$(0%), to indicate the old format quotas.
5-6	Reserved; should be 0.
7-8	PPN. Zero for both values means the current account.
9-12	New password in Radix-50 format (pre-V9.0 format). All zeros means no change. See the next section, "Set Password," for a description of how to set passwords using V9.0 format.
13-14	CHR$(N%)+CHR$(SWAP%(N%)), where N% is the number of blocks for the quota. If N% is zero, the quota is unlimited (see byte 21).
15-16	Expiration date, in RSTS/E internal format: (day of year) + [(number of years since 1970) * 1000]. CHR$(0%) to indicate no change.
17-20	Reserved; should be 0.
21	CHR$(255%) to change the logged-out disk quota (see bytes 13-14). CHR$(0%) to indicate that no change is to be made to the quota.
22	Reserved; should be 0.
23-24+	Device name. If no device name is specified, SY: is used.
25+	Unit number.
26+	Unit number flag.
27-28	Must be CHR$(0%).
29-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

GACNT Change quota or password within the group

WACNT Change any quota or password

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME You specified a password that was less than six characters. Or, you specified a password that contained illegal characters.	2
?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

You can use this call to perform any or all of the following operations:

- Change logged-out disk quotas for the account using the old format.
- Change a user's password using the old format. To change a password using the new 14 character ASCII format, see the next section, "Set Password."
- Change the expiration date for the account.

Bytes 13-14 specify the logged-out disk quota. Quota values must be in the range 0 to 65535. A value of 0 means unlimited; for RDS1.1 disks, this is converted into 16777215 (2^24-1).

Set Password (New Format)

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(8%), the change quota, password, kill job, and disable terminal code.
3-4	Reserved; should be 0.
5-6+	PPN. Zero for both values means the current account.
7-20	New password as 14 bytes of ASCII data (V9.0 format), padded with nulls if necessary. If the first byte is 0, the password information is deleted, signifying an account that cannot be logged in to in any way. See Discussion.
21-22	Reserved; should be 0.
23-24+	Device name. If no device name is specified, SY: is used.
25+	Unit number.
26+	Unit number flag.
27	Must be CHR$(255%).
28	Must be CHR$(0%).
29-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

GACNT	Change any password within the group
WACNT	Change any password

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME You specified a password that was less than six characters. Or, you specified a password that contained illegal characters (such as ?).	2
?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

You can use this call to set a 14 character ASCII password for an account. Passwords can contain any printing character except question mark (?), including punctuation and supplemental characters. The system treats lower- and uppercase characters as equivalent. If the account is set up to have a readable password (/LOOKUP qualifier on the SET ACCOUNT or CREATE/ACCOUNT command), then the password must be 6 characters long and can contain only letters and digits.

You specify the new password in bytes 7-20. If the first byte is 0, the system deletes the password information, in effect changing the account into a no-user account. See the *RSTS/E System Manager's Guide* for more information on accounts.

Kill Job
F0=8

Kill Job

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(8%), the change password/quota, kill job, and disable terminal code.
3	CHR$(N%), where N% is the number of the job to kill, or 0 to kill the caller's job.
4-26	Reserved; should be 0.
27	Must be CHR$(0%); this byte differentiates the kill job call from the disable terminal call.
28	Must be CHR$(255%).
29-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

JOBCTL

Possible Errors

Meaning	ERR Value
?ILLEGAL SYS() USAGE The job number specified is invalid.	18

Discussion

This call provides the normal way for a privileged job to terminate itself under programmed control. The job must execute the Kill a Job SYS call with job number specified as 0. The kill does all of the cleanup that the Logout SYS call (SYS 5) does and can be executed under program control by any privileged program.

Note that if you do not have JOBCTL privilege, you use the Logout call instead of this call to kill your current job. See SYS call 5, Logout.

Disable Terminal

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(8%), the change password/quota, kill job, and disable terminal code.
3	CHR$(N%), where N% is the keyboard number of the terminal to disable.
4-26	Reserved; should be 0.
27	Must be CHR$(255%) to differentiate this call from the kill job call.
28	Must be CHR$(255%).
29-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

HWCTL

Possible Errors

Meaning	ERR Value
?ILLEGAL SYS() USAGE	18

Keyboard number is greater than the number of terminals on the system; keyboard number corresponds to a line used by a pseudo keyboard; or the terminal is currently opened or assigned by a job.

Discussion

This SYS call disables a keyboard line. After the system executes this function, it does not process or echo input from the disabled keyboard. The system also ignores any output for the disabled keyboard. Once a keyboard is disabled, it remains disabled until the next time-sharing session is started or the line is reenabled with the Set System Defaults SYS call (SYS 34).

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

To disable a terminal (other than KB0:) for more than one time-sharing session, use the DCL SET command (see the *RSTS/E System Manager's Guide*).

Return Error Message
F0=9 (UU.ERR)

Return Error Message

Data Passed

Bytes	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	Reserved; should be 0.

Data Returned

Bytes	Meaning
1	The current job number times 2.
2	If job is attached, current keyboard number times 2 of terminal to which job is attached. If job is 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.

Privileges Required

None.

Possible Errors

None.

Discussion

This 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 use of the call is to print the system header line containing the system name and the local installation name. To do this, use the character representation of the ERR value of CHR$(0%) in the call.

The following sample program extracts and prints the message associated with an error number that you supply:

```
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

RUNNH
ERROR NUMBER? 0
RSTS V9.0 SYSTEM #880
```

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 null characters. The first character of the text (except for message number 0) is the severity indication. See Appendix C for more information.

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

Allocate Device/Assign User Logical
F0=10 (UU.ASS)

Allocate Device/Assign User Logical

This call has two subfunctions:

- Allocate/Reallocate Device
- Assign User Logical

Allocate/Reallocate Device

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(10%), the allocate/reallocate device and assign user logical code.
3-4	Reserved; should be 0.
5-6	If bytes 7 through 10 are 0, these bytes contain the assignable PPN (@). If bytes 7 through 10 contain a logical device name, these bytes contain the PPN assigned to that logical device.
7-10	To allocate a device, bytes 7 through 10 must be 0. To reallocate a device, byte 7 is the job number to which the device is reallocated. Bytes 8 through 10 must be 0.
11-12+	Either DOS or ANS (in Radix-50 format) to specify DOS or ANSI label format for the magnetic tape drive.
13-16	Reserved; should be 0.
17-18	CVT%$(SWAP%(-32767%)), to allocate a device that is currently allocated to another user. This use requires HWCTL privilege and requires that the target device not be open. If you do not want this operation, bytes 17 and 18 are 0.
19-22	Reserved; should be 0.
23-24+	Device name.

25+	Unit number.
26+	Unit number flag.
27-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1-2	Not used.
3	The job number of the previous owner of the device. A value of 0 indicates the device was not previously allocated.
4-30	Not used.

Privileges Required

DEVICE	Allocate a restricted device
HWCTL	Reallocate a device to a job in another account, or seize a device

Possible Errors

Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE During a reallocate call, the specified device is currently open or has an open file.	3
?NOT A VALID DEVICE The device name specified in bytes 23 and 24 is a logical device name for which a physical device is currently not assigned.	6
?DEVICE NOT AVAILABLE The device specified in bytes 23 through 26 exists on the system but the operation fails for one for the following reasons: o The device is currently reserved by another job (see bytes 17 and 18). o The user does not have sufficient privilege to own the device. For example, a user without HWCTL privilege tried to allocate a device that	8

is currently allocated to another user.

- The device or its controller is disabled.
- The device is a keyboard line for a pseudo keyboard only.

?PROTECTION VIOLATION 10
You do not have sufficient privilege to perform either of these operations:

- Allocate or reallocate a restricted device.
- Reallocate a device to a job that is logged in to an account other than your current account.

?ILLEGAL NUMBER 52
An attempt is made to transfer control to a nonexistent job. This error can occur only during a reallocate call.

Discussion

The Allocate/Reallocate call uses bytes 17 and 18 to allocate or reallocate a device that is currently allocated by another job. For the call to be successful, the caller must have HWCTL privilege, the target device must not be open, and the current owner cannot be performing a directory operation on that device.

The allocate call reserves a physical device to a job; the reallocate transfers assignment of a currently owned device to another job. (The SET DEVICE/RESTRICT command designates that certain devices are restricted and therefore require DEVICE privilege to be allocated.) The action is equivalent to the DCL ALLOCATE command (see the *RSTS/E System User's Guide*). Users without HWCTL privilege can only reallocate a device to a job running under the same PPN as the caller.

Example

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

Assign User Logical

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(10%), the assign/reassign device and assign user logical code.
3-4	Reserved; should be 0.
5-6+	The PPN to be assigned.
7-10	The logical device name (in Radix-50 format) to be assigned.
11-20	Reserved; should be 0.
21	CHR$(255%) to enable protection code assignment (see byte 22).
22	The protection code to be assigned. Byte 21 must be 255.
23-24+	Device name.
25+	Device Unit number.
26+	Unit number flag.
27-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

None.

Possible Errors

Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE During an assign call, more than four user logical assignments are made (three, if you use a PPN).	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

Discussion

This call assigns logical device names, logical PPNs, 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 PPN, specify the number in bytes 5 and 6. To assign a user default protection code, specify the code in bytes 21 and 22.

Deallocate a Device or Deassign User Logical

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(11%), the code to deallocate a device or deassign user logical.
3-4	Reserved; should be 0.
5-6	For user logical, -1 to deassign the default PPN. For device deallocation, must be 0.
7-10	For user logical, the logical device name (in Radix-50 format) to be removed. Otherwise, must be 0.
11-20	Reserved; should be 0.
21	For user logical, CHR$(255%) to enable protection code removal. Otherwise, must be 0.
22	Reserved; should be 0.
23-24	For device deallocation, the device name to be deallocated. For user logical, must be 0.
25	For device deallocation, the device unit number to be deallocated. For user logical, must be 0.
26	For device deallocation, the unit number flag. For user logical, must be 0.
27-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

None.

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 deallocation calls.	6

Discussion

This call deassigns logical device names, logical PPNs, and default output protection codes. To deassign a logical device name, specify the name in bytes 7 through 10. To deassign a PPN, specify the number in bytes 5 and 6. To deassign a user logical protection code, specify 255 in byte 21. Note that if these bytes do not contain specific deassignments, the call deassigns all user logical device names, PPN, and protection code assignments.

The Deallocate a Device call performs the same action as the DEALLOCATE DCL command (see the *RSTS/E System User's Guide*). For example, the following statement deallocates line printer unit 1, which is allocated to the current job:

```
10  A$ = SYS(CHR$(6%)+CHR$(11%)+STRING$(20%,0%)+
         "LP"+CHR$(1%)+CHR$(255%))
         ! DEALLOCATE LP1:
```

Deallocate All Devices
F0=12 (UU.DAL)

Deallocate All Devices

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(12%), the deallocate all devices code.
3-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

None.

Possible Errors

None.

Example

The following statement deallocates all devices currently allocated to the job:

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

Zero a Device

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(13%), the zero a device code.
3	For disk, CHR$(N%), where N% determines the action taken on the files and the UFD: 0% Delete all files except those write-protected against owner; retain UFD. 255% Delete all files regardless of their protection codes; delete UFD (requires GACNT or WACNT privilege). For magnetic tape and DECtape, set this byte to 0.
4	Reserved; should be 0.
5-6+	PPN. See Discussion.
7-10+	Volume ID, in two Radix-50 words, for volume label (ANSI format magnetic tape only).
11-22	Reserved; should be 0.
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	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Zero a Device
F0=13

Privileges Required

None	Zero your own account if it does not reside on a restricted device
GWRITE	Zero any account in the group
WWRITE	Zero any account
SYSIO	Zero any account in group [0,*] (with WWRITE)
GACNT	Deallocate UFD of any account in the group
WACNT	Deallocate UFD of any account
DEVICE	Access restricted devices

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.	5
?NOT A VALID DEVICE The device or its type specified in bytes 23 through 26 is not configured on the system.	6
?DEVICE NOT AVAILABLE The 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, or the device or its controller has been disabled by the system manager.	8
?PROTECTION VIOLATION You attempted to zero an account other than your own without sufficient privilege.	10
?ILLEGAL SYS() USAGE Bytes 5 and 6 do not contain a valid PPN.	18

?DEVICE NOT FILE STRUCTURED 30
The specified device does not allow access by file name.

This call also returns device-dependent errors such as ?Disk pack not mounted (ERR=21) and ?Magtape select error (ERR=39).

Discussion

This call zeros DECtape, magnetic tape, or disk.

For DECtape or magnetic tape, this call zeros the entire medium. On DECtape, this call also clears the directory. The section "Initializing Magnetic Tapes," in Appendix A, describes what actions occur when magnetic tape is zeroed.

For disk, you can delete files in a directory. If you do not have sufficient privilege you can specify only the current directory, and you cannot delete the UFD. Furthermore, if your account resides on a restricted disk, you must have the DEVICE privilege.

If you have GACNT privilege, you can delete files and the UFD for any account in your group. If you have WACNT privilege, you can delete the files and the UFD of any account. Note that this call does not delete accounts. To delete an account, use the Delete User Account call, SYS 1.

Bytes 5 and 6 must contain a valid PPN. The system returns the error ?Illegal SYS() usage (ERR = 18) if bytes 5 and 6 are zero.

The value in byte 3 determines what happens to the files and the UFD. A value of 0% in byte 3 deletes files in a directory that are not write-protected against the owner and retains the UFD. A value of 255% in byte 3 deletes all files in a directory, regardless of protection code. In addition, if you have the appropriate GACNT or WACNT privilege, the value 255% in byte 3 deletes the UFD (deallocates the disk space that was allocated to the UFD). The UFD is always retained for insufficiently privileged users.

Note that you should not normally use the option to delete the UFD. When the account is created, the UFD is often placed in a specific location on the disk to optimize disk performance. Deleting the UFD causes any prior placement to be lost.

Zero a Device
F0=13

Example

```
10    P0%=10%
      \P1%=20%
20    A$=SYS(CHR$(6%)+CHR$(13%)+STRING$(2%,0%)+
         CHR$(P1%)+CHR$(P0%)+STRING$(24%,0%))
         ! ZERO [10,20] ON THE SYSTEM.
         ! IF NONPRIVILEGED, CURRENT ACCOUNT
         ! MUST BE [10,20]
         ! UFD AND ANY FILES WRITE-PROTECTED
         ! AGAINST OWNER ARE RETAINED.
30    A$=SYS(CHR$(6%)+CHR$(13%)+STRING$(20%,0%)+
         "MT"+CVT%$(0%))
         ! ZERO MT:
```

Read/Read and Reset Accounting Data

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(14%), the read or read and reset accounting data code.
3-4	CHR$(I%)+CHR$(SWAP%(I%)), where I% is the index number of the account to read. I% can be one of the following values: 0% — Read the account specified in bytes 7 and 8. Nonzero — Search for an account based either on this index entry or on a wildcard PPN search. The value of W% in byte 9 controls which function the call performs. See the table in the Discussion for a summary of legal byte values.
5-6	CHR$(R%), where R% can be one of the following values: 0% — Indicates read-only. Nonzero — Indicates read and reset. See the Discussion for a list of the accounting data that gets reset. If the job executing this call does not have the appropriate privileges, the system does not access this word and performs only a read operation.
7-8	PPN. If bytes 7 and 8 are 0%, data for the current account is returned. If W% in byte 9 is 2%, bytes 7 and 8 can be 255% to indicate that the PPN contains wildcards. See the section "Project-Programmer Number" for a description of each byte; see the table in the Discussion for a summary of legal values.
9	CHR$(D%+W%+Q%+P%), where: D% indicates whether to return the number of blocks owned by the account: D%=0% — Call returns number of blocks used. D%=1% — Call does not return this data.

W% indicates whether the PPN in bytes 7 and 8 contains wildcards:

W%=0% PPN corresponds to a real account.

W%=2% PPN contains a wildcard.

Q% indicates the type of accounting information to return:

Q%=0% Call returns general accounting information.

Q%=4% Call returns disk quota information.

P% indicates whether to return a protection violation if a caller attempts to perform a function without sufficient privilege:

P%=0% Do not perform a privilege check. The call performs a read-only lookup on the caller's PPN.

P%=8% Perform a privilege check and return the error ?Protection violation (ERR=10) if the caller does not have sufficient privilege.

10-22 Reserved; should be 0.

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 Reserved; should be 0.

Data Returned when Q% in byte 9 = 0% (accounting information format)

Bytes	Meaning
1	The current job number times 2.
2	The number of clusters used to store the User File Directory (UFD).
3-4	Same as bytes 3-4 in data passed.
5-6	Number of blocks used. The maximum number returned is 65535 blocks. If more than 65535 blocks are in use, only 65535 is returned. The quota information is accurately returned in the quota information format of this call.
7-8	PPN of the account read.
9-12	Password of the account read, in Radix-50 format. If the password is marked as not readable (/NOLOOKUP), 0 is returned. This data is returned only if the caller has the appropriate privileges (GACNT for group; WACNT for all).
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	Logged-out disk quota in number of blocks; 0 means unlimited quota. This value is 16 bits. 24 bit quotas are returned by the quota information format of this call.
29	User file directory cluster size.
30	Not used.

Data Returned when Q% in byte 9 = 4% (quota information format)

Bytes	Meaning
1	The current job number times 2.
2	Reserved.
3-4	Same as bytes 3-4 in data passed.
5-6	Reserved.
7-8	PPN of the account read.
9-10	Logged-out quota (LSB).
11-12	Logged-in quota (LSB).
13	Logged-in quota (MSB).
14	Logged-out quota (MSB).
15	Reserved.
16	Current usage (MSB).
17-18	Reserved.
19-20	Current usage (LSB).
21-22	Count of open files, and logged in jobs in that account. Open files = low 10 bits, number of jobs = high 6 bits.
23-26	Device name and number as passed.
27-30	Reserved.

Privileges Required

None	Read information for your own account
GACNT	Read or read and reset information for accounts in the group
WACNT	Read or read and reset information for all accounts

Possible Errors

Meaning	ERR value
?BAD DIRECTORY FOR DEVICE The account does not have all the necessary directory structures.	1
?CAN'T FIND FILE OR ACCOUNT The PPN specified does not exist on the disk, the index specified is greater than the number of accounts on the disk, or you specified an illegal combination of values in bytes 3-4, bytes 7-8, and byte 9.	5
?PROTECTION VIOLATION The caller does not have sufficient privilege to perform the indicated function.	10
?ILLEGAL SYS() USAGE Device specified is not a disk.	18

Discussion

This SYS call performs the following functions:

- Looks up accounts on a disk and reads accounting data. You can:
  - -- Specify an individual account
  - -- Specify a search based on an index number
  - -- Specify a search based on a wildcard PPN match
- Looks up accounts on a disk and reads and resets accounting data. You can use the same three methods to control the account search. This call resets the following accounting data to zero:
  - -- CPU time
  - -- Connect time
  - -- Kilo-core ticks
  - -- Device time
- Returns information about the number of blocks used and the logged-out disk quota. If you are using the new format disk quotas feature (V9.0), this call returns information about the logged-in and logged-out quotas, as well as the current usage.

Note that this system function call does not always perform a password lookup. This affects programs that currently look up passwords for the purpose of logging in to a given account. These programs should use the "spawn logged in" subfunction of the Create a Job SYS call (SYS 24).

Bytes 3-4, bytes 7-8, and byte 9 control how the call looks up accounts. Table 7-7 lists the legal combinations of byte values and their corresponding results. Any other combination of values returns the error ?Can't find file or account (ERR=5).

Table 7-7: SYS 14 Legal Byte Value Combinations

Bytes 3-4 I%	Byte 9 W%	Bytes 7-8 PPN	Account Accessed
0	0	0	Current account
0	0	PPN	Account specified by PPN
Nonzero	0	Forced to [255,255]	The 'Ith' account, where I is the value specified in bytes 3-4
Nonzero	2	[255,y]	The 'Ith' account matching [*,y], where I is the value specified in bytes 3-4
		[x,255]	The 'Ith' account matching [x,*], where I is the value specified in bytes 3-4
		[255,255]	The 'Ith' account, where I is the value specified in bytes 3-4

If you want to look up all accounts on a disk, start the index (bytes 3 and 4) at 1 and increment it for each call.

Note that this call works differently for users without the appropriate GACNT or WACNT privilege. If a job without sufficient privilege executes this call, the system forces bytes 3 through 8 in the data passed to the values shown:

Byte	Value	Action
3-4	0%	Look up the account specified in bytes 7 and 8.

5-6	0%	Read-only.
7-8	current PPN	Look up data for current PPN.

A job with the appropriate privileges can both read and reset accounting data. GACNT privilege grants access to any account in the group. WACNT privilege grants access to all accounts.

You can instruct the call to perform a privilege check by setting P% in byte 9. This allows a program to make sure that the system is performing the proper function, rather than just performing a lookup on the caller's account.

If an appropriately privileged job executes this call and bytes 5 and 6 of the data passed are nonzero, the following account information is read and reset to zero:

- CPU time
- Kilo-core ticks
- Connect time
- Device time

Because the system must scan the entire directory to return the number of blocks owned by an account, significant extra processing time is required to obtain usage information. This applies to RDS1.1 and RDS0.0 format disks. If you do not need this information, specify a value of 1 for D% in byte 9 to speed up the execution of this call. For RDS1.2 format disks, no time is saved by using this option.

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. Figure 7-3 is a graphic representation.

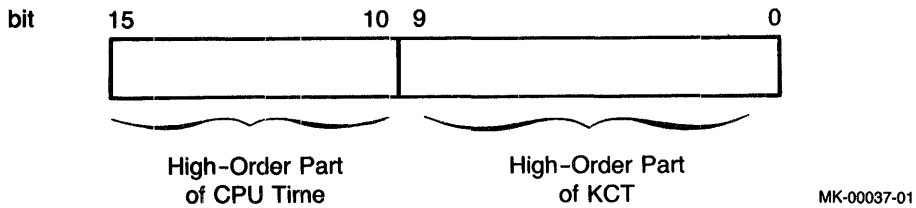


Figure 7-3: High-Order Bits of CPU Time and KCTs

Directory Lookup

This section describes the SYS calls that look up file specifications under program control. Although only two codes are available, four different operations are possible:

- Directory lookup on index (SYS 15)
- Special magnetic tape directory lookup (SYS 15)
- Disk directory lookup by file name (SYS 17)
- Disk wildcard directory lookup (SYS 17)

As a result, four descriptions appear in this section.

The four operations return data in the following format:

Data Returned

Bytes	Meaning
1	The current job number times 2.
2	Not used.
3-4	Same as bytes 3-4 in data passed.
5-6	PPN (if applicable) of the file read. For magnetic tape, these bytes return the value passed.
7-10	File name in Radix-50 format. See the section "Unpacking the Returned Data," earlier in this chapter for a description of converting a string in Radix-50 format.
11-12	File type in Radix-50 format.
13-14	Length in blocks. Not used for Special Magnetic Tape Directory Lookup call (SYS 15); Least Significant Bits (LSB) for files larger than 65535 blocks.
15	Protection code of the file.
16	The Most Significant Bits (MSB) of the file size. If a nonzero number is returned, it indicates that the file size is greater than 65535 blocks. (The Special Magtape Directory Lookup call (SYS 15) does not return this information.)

17-18 For disk, the date of last access; for DECtape and magnetic tape, returns the date of creation. (Note that 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.)

19-20 The date of creation for disk.

21-22 The time of creation for disk; for DOS magnetic tape, the PPN.

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 for SYS 17.)

30 The USTAT byte from the UFD Name entry (returned for disk only).

This byte contains the following internal flag information:

Value	Meaning
1%	Reserved.
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%	Reserved.
128%	File is marked for deletion.

Keep this information in mind when you use the Directory Lookup calls:

- If you specify either DECtape or magnetic tape, the monitor allocates the related unit to the calling job while the call executes. The unit remains allocated after the call completes.
- When you repeatedly execute one of the calls on disk and increment the index for each repetition, the execution time

increases for each successive call. The increase occurs because the monitor must read the file name blocks for indexes numbered 0 through N-1 before it reads the file name 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 file name block. Note that the index scheme for SYS call 16 (and 25) differs from that of SYS call 14.

The Open-Next SYS call (SYS 33), on the other hand, has constant execution time throughout a directory, because it uses an I/O channel to keep context information.

- When you repeatedly execute 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. Because the monitor cannot determine how many files exist on each unit of a multiple public disk structure, it must read the file name blocks of each unit beginning at unit 0 until the Nth file is read. Therefore, on such a system, you can decrease the execution time by executing 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:).

  The Open-Next SYS call (SYS 33) does not have this limitation.

Directory Lookup on Index

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(15%), the directory lookup on index code.
3-4	CHR$(N%)+CHR$(SWAP%(N%)), where N% is the index of the file to read. If N% is 0, the call returns the data for the first file in the directory. If N% is x, (some nonzero value), the call returns 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 the Special Magnetic Tape Directory Lookup SYS call (SYS 15) 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	PPN 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 the section, Project-Programmer Number, for a description of these bytes.
7-16	Reserved; should be 0.
17-18	CHR$(M%)+CHR$(SWAP%(M%)), where M% indicates whether to return information about files that are marked-for-delete: M%=0% Skip marked-for-delete files. M%=16384% Return information about marked-for-delete files.
18-22	Reserved; should be 0.
23-24+	Device name to look up. If both bytes are 0, SY: (the public structure) is used.
25+	Unit number.
26+	Unit number flag.
27-30	Reserved; should be 0.

Directory Lookup on Index
F0=15

Data Returned

See the introductory section, "Directory Lookup," for a description of the Data Returned.

Privileges Required

None	Look up files in your own directory, or in the directory of another account to which you have read or execute access
GREAD	Look up files in the directory of any account in the group
WREAD	Look up files in the directory of any account
DEVICE	Access restricted devices

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

The call also returns device-dependent errors such as ?Device hung or write locked (ERR=14) and ?Disk pack not mounted (ERR=21).

Discussion

This call returns directory information on a file. The DCL DIRECTORY command uses the same routines as this call to print a directory listing. The order of the files in the listing is by index value from the lowest to the highest. You can therefore determine the index value for a file by counting its position in a DIRECTORY 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.

Special Magnetic Tape Directory Lookup

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(15%), the directory lookup on index code.
3-4	CHR$(N%)+CHR$(SWAP%(N%)), where N% is the index of the file to read. If N% is 0, the call rewinds the tape and returns the data for the first file in the directory. If N% is some nonzero value, the call returns the data for the next file on the tape. See Discussion.
5-6	Both bytes are CHR$(255%) to execute the special magnetic tape directory lookup.
7-22	Reserved; should be 0.
23-24	Device name, must be a magnetic tape (not DECtape).
25+	Unit number.
26+	Unit number flag.
27-30	Reserved; should be 0.

Data Returned

See the introductory section, "Directory Lookup," for a description of the Data Returned.

Privileges Required

DEVICE	Access restricted devices

Possible Errors

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT No more files exist on the tape.	5

?DEVICE NOT FILE STRUCTURED 30
The device specified in bytes 23 and 24 is not file structured.

Discussion

The standard Directory Lookup on Index call (SYS 15) executed on a magnetic tape unit causes the monitor to:

1. Read one record from the tape (a label record). 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.

2. Space the tape forward to the next end-of-file (EOF) or end-of-volume (EOV) record and calculate the number of records in the file.

3. Return the directory information if the account number of the file matches the one specified in the call or if both bytes in the account specification in the call are zero.

When the monitor executes the action that step 1 describes, you must position the tape immediately before a label record. Otherwise, the operation generates an error or returns incorrect information.

In an application program that must search a tape for a specific file and read each specific file found, the OPEN FOR INPUT statement necessitates a rewind operation. When you execute an OPEN FOR INPUT statement on a file-structured magnetic tape, the monitor:

1. Reads one record from the tape (must be a label record).

   - If the read operation is successful, opens the file and returns control to the user program.

   - If the read operation is unsuccessful and this is the first label read, rewinds the tape and executes the action that step 1 describes.

2. Returns an error if it detects the logical end-of-tape.

3. Skips to the end of the file and executes the action in step 1 if the label read does not match.

The required rewind operations consume time. To avoid the rewind operations, you can execute the special magnetic tape directory lookup call and perform certain actions. By specifying both bytes 5 and 6 as

CHR$(255%) in the call, you cause the monitor to:

1. Read a record from the tape, which must be a label record.

2. Backspace one record, which leaves the tape in a position to read the label record again.

3. Return the directory information (except for file length) to the program.

To take advantage of these special actions, you must determine from the information returned whether the file is the one required. Then follow this procedure:

1. If the file is the one required, execute the OPEN FOR INPUT statement using the file name and requesting no rewind. The action executes without a rewind because the tape is positioned properly.

   If the file is not required, issue the normal form of the directory lookup function (Directory Lookup on Index, SYS 15) with a nonzero index value in bytes 3-4. This causes the file to be skipped properly on either DOS or ANSI format tape. See Chapter 2 for more information.

2. 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 Magnetic Tape Directory Lookup 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.

Disk Directory Lookup by File Name

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(17%), the disk directory lookup by file name and disk wildcard directory lookup code. See the next section for a description of the Disk Wildcard Directory Lookup call.
3-4	Both bytes must be CHR$(255%).
5-6+	PPN of the file to look up. If both bytes are 0, the current account is used.
7-10+	File name in Radix-50 format.
11-12+	File type in Radix-50 format.
13-22	Reserved; should be 0.
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	Reserved; should be 0.

Data Returned

See the introductory section, "Directory Lookup," for a description of the Data Returned.

The following bytes differ for this call.

Bytes	Meaning
23-26	If a name of the public structure, such as SY: or DK:, is passed, or if no device name is passed, then the actual device where the file resides (for example, DB2:) is returned.
29-30	The file identification index is returned.

Privileges Required

None	Look up files in your own directory, or in the directory of another account to which you have read or execute access
GREAD	Look up files in the directory of any account in the group
WREAD	Look up files in the directory of any account
DEVICE	Access restricted devices

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME File name in bytes 7 through 10 is missing.	2
?CAN'T FIND FILE OR ACCOUNT The device specified in bytes 23 and 24 is not a disk, or the file specified does not exist on the specified disk. This error also occurs when the user does not have sufficient privilege to read the specified file.	5

Discussion

This call works only on disk files and returns directory information for the specified file. If you try to look up a file in some other account, you must have read or execute access to the file for the lookup to succeed. If you do not have access rights, the call returns the error ?Can't find file or account (ERR=5).

Disk Wildcard Directory Lookup

Data Passed

Same as that described in the previous section, "Disk Directory Lookup by File Name," except for:

Bytes	Meaning
3-4	CHR$(I%) + CHR$(SWAP%(I%)). If I% is 0, the call returns the data for the first file that matches the wildcard specification. If I% is x (some nonzero value), the call returns the data for the x + 1 file that matches the wildcard specification.
7-10+	Radix-50 representation of a wildcard file name specification where an * character can replace the file name or a ? character can replace any character in the file name. Used with the file type in bytes 11 and 12 to create a wildcard file specification.
11-12+	Radix-50 representation of a wildcard file type specification, where an asterisk (*) character can replace the file type or a question mark (?) character can replace any character in the file type. Used with the file name in bytes 7 through 10 to create a wildcard file specification.
17-18	CHR$(M%)+CHR$(SWAP%(M%)), where M% indicates whether to return information about files that are marked-for-delete: M%=0% Skip marked-for-delete files. M%=16384% Return information about marked-for-delete files.

Data Returned

See the introductory section, "Directory Lookup," for a description of the Data Returned.

Privileges Required

None	Look up files in your own directory, or in the directory of another account to which you have read or execute access
GREAD	Look up files in the directory of any account in the group

WREAD Look up files in the directory of any account

DEVICE Access restricted devices

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME No file name appears in bytes 7 through 10.	2
?CAN'T FIND FILE OR ACCOUNT The device specified in bytes 23 and 24 is not a disk, or no match exists for the index value given in bytes 3 and 4.	5
?DEVICE IS RESTRICTED The disk is in the restricted state and the job does not have DEVICE privilege.	22

Discussion

This call allows a program to supply a wildcard specification and to increment an index value to get directory information for all occurrences of files matching the wildcard specification. The following are typical wildcard specifications:

Specification	Meaning
FILE??.*	All files with FILE as the first four characters in the name and with any file type (including no file type).
*.BAS	All files with .BAS file types.
*.BA?	All files with BA as the first two characters in the file type.

The program supplies an index of 0 and executes the call. The system returns directory information for the first file that 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 the error ?Can't find file or account (ERR=5) to indicate no more matching occurrences exist in the account. The entire procedure relieves the program of the overhead required to translate each file name in the directory and to compare for a match.

Set Terminal Characteristics

This call has two subfunctions:

- Set Terminal Characteristics - Part I
- Set Terminal Characteristics - Part II

To use this call, you need to understand the concept of terminal ownership. A terminal is "owned" when:

- It becomes attached to a job by logging in. When data is entered at a free terminal, the system starts a job to handle the input and gives the job the next available job number. The system then starts the LOGIN program to allow the user to log in to the system. (See the *RSTS/E System User's Guide* for the operational details.)

  When a user is logged in to the system, the system associates the activated job with both the terminal at which the user is typing and the account number used for system identification. The job is then considered active on the system and in attached mode (or attached to the terminal). The system associates I/O channel 0 with the terminal that activated the job. The terminal associated with channel 0 is called the job's console terminal or console keyboard. A job can have only one console terminal, the keyboard to which it is attached.

- It is opened on a nonzero channel. A job can own several terminals that are open on nonzero channels.

- It is allocated for the use of a job (with the the DCL ALLOCATE command).

Set Terminal Characteristics - Part I

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(16%), the set terminal characteristics code.
3	CHR$(0%), the code to specify part I of the SYS call.
4	CHR$(N%), where N% is 255% for the current keyboard (requires no privilege) or is the keyboard number of the terminal to alter (requires HWCFG privilege).
5	CHR$(N%), where N% is 0% for no change or is the terminal width plus 1. The call sets the number of characters for each line to N%-1, where N% is in the range 2% to 255%. The WIDTH n command sets this byte.
6	CHR$(N%), where N% is: 0% No change. 128% Enable the hardware horizontal tab feature. The /TAB qualifier of the SET TERMINAL command sets this characteristic. (The device must have the necessary hardware.) 255% Enable software horizontal tab positions, which are set every 8 character positions beginning at position 1. The /NOTAB qualifier of the SET TERMINAL command sets this characteristic.
7	CHR$(N%), where N% is: 0% No change. 128% Enable the software to perform form feed and vertical tab operations by executing four line feed operations. The /NOFORM_FEED qualifier of the SET TERMINAL command sets this characteristic. 255% Enable hardware form feed and vertical tab. The /FORM_FEED qualifier of the SET TERMINAL command sets this characteristic. (The device must have the necessary hardware.)

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

0% No change.

128% Allow the terminal to receive and print lowercase characters (CHR$(96%) through CHR$(126%)). The /LOWERCASE=OUTPUT qualifier of the SET TERMINAL command sets this characteristic.

255% Have the system translate lowercase characters to uppercase before transmitting to a terminal. The /UPPERCASE=OUTPUT qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

128% Have the terminal not respond to XON CHR$(17%) and XOFF CHR$(19%) characters because it lacks the necessary hardware. The /NOHOST_SYNC qualifier of the SET TERMINAL command sets this characteristic.

255% The terminal has the necessary 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 /HOST_SYNC qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

128% 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 /NOLOCAL_ECHO qualifier of the SET TERMINAL command sets this characteristic.

255% Have the terminal (or its modem) locally echo the characters typed. The computer does not echo the characters received. The /LOCAL_ECHO qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

128% The terminal does not have features of a video display terminal. The /HARDCOPY qualifier of the SET TERMINAL command sets this characteristic. Specifying this value causes byte 17 to be set to 128 (/NOTTSYNC).

255% The terminal is a video display, or cathode ray tube (CRT), and uses the following features:

- Responds to the synchronization protocol described by byte 17.
- The system executes a DELETE character by sending a backspace, a space, and a backspace to the terminal.
- Any location on the screen can be addressed by direct cursor placement.

The /SCOPE qualifier of the SET TERMINAL command sets this characteristic. Specifying this value causes byte 17 to be set to 255 (/TTSYNC).

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

0% No change.

128% The system treats certain characters that it receives as follows:

- Translate CHR$(125%) and CHR$(126%) into the ESC character CHR$(27%) (unless the /NOALT_MODE characteristic is subsequently set).
- Translate lowercase characters (CHR$(64%) through CHR$(94%)) to uppercase equivalents (CHR$(96%) through CHR$(126%)).

The /UPPERCASE=INPUT qualifier of the SET TERMINAL command sets this characteristic.

255% The terminal transmits the full ASCII character set and the system treats special characters as follows:

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

- Treat CHR$(125%) and CHR$(126%) as printed characters } and ~.
- Do not translate lowercase characters to uppercase format.

The /LOWERCASE=INPUT qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

1% No fill factor for the terminal. The /NOCRFILL qualifier of the SET TERMINAL command sets this characteristic.

n% Set fill factor of the terminal to N%-1. The /CRFILL=n qualifier of the SET TERMINAL command sets this characteristic.

255% Reserved.

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

0% 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 (requires HWCFG privilege). Note that you can set internal speed value only for DH11, DZ11/DZV11/DZQ11, and DHV11/DHU11 interface lines as shown in Table 7-8. The /SPEED qualifier of the SET TERMINAL command sets this characteristic.

Table 7-8: Internal Speed Values for Terminal Interface Lines

DH11:		DZ11/DZV11/DZQ11:		DHV11/DHU11:	
Code	Speed	Code	Speed	Code	Speed
1	0	1	0	1	0
2	50	2	50	2	75
3	75	3	75	3	110
4	110	4	110	4	134.5
5	134.5	5	134.5	5	150
6	150	6	150	6	300
7	200	7	300	7	600
8	300	8	600	8	1200
9	600	9	1200	9	1800
10	1200	10	1800	10	2000
11	1800	11	2000	11	2400
12	2400	12	2400	12	4800
13	4800	13	3600	13	Reserved
14	9600	14	4800	14	9600
		15	7200	15	19200
		16	9600		

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

0% No change.

1% Do not set the output parity bit. The /NOPARITY qualifier of the SET TERMINAL command sets this characteristic.

2% Generate the output parity bit for even parity format. The /PARITY=EVEN qualifier of the SET TERMINAL command sets this characteristic.

3% Generate an output parity bit for odd parity format. The /PARITY=ODD qualifier of the SET TERMINAL command sets this characteristic.

4% Inhibit altering of the data length. The data length is the number of data bits (not counting start, stop, or parity bits) transmitted on the line. See Discussion.

16% Turn off the 8-bit characteristic (7 bits). The /NOEIGHT_BIT qualifier of the SET TERMINAL command sets this characteristic. See Discussion.

24% Set the 8-bit characteristic. The /EIGHT_BIT qualifier of the SET TERMINAL command sets this characteristic. See Discussion.

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

0% Both the receive (input) and transmit (output) speeds are determined by the value n in byte 14. The /SPEED qualifier of the SET TERMINAL command sets this characteristic.

n% The internal speed value to determine the baud rate at which the terminal transmits characters when a split speed setting is used (requires HWCFG privilege). You can use split speed settings with the DH11 interface line only. The /SPEED=(input[,output)] qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

128% The terminal ignores the synchronization protocol that is described in the following 255 value. The /HARDCOPY and /NOTTSYNC qualifiers of the SET TERMINAL command set this characteristic. In addition, the system automatically sets this value when byte 11 is 128.

255% The terminal obeys the synchronization protocol:

- The computer stops sending characters if the terminal transmits a CHR$(19%) character (XOFF, or the CTRL/S combination).
- The computer resumes sending characters when the terminal transmits a CHR$(17%) character (XON, or the CTRL/Q combination).

The /SCOPE and /TTSYNC qualifiers of the SET TERMINAL command set this characteristic. In addition, the system automatically sets this value when byte 11 is 255.

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

0% No change.

128% The system prints a control character as the up arrow or circumflex character (↑ or ^) followed by the equivalent printable character. For example, the CTRL/D combination is printed as ^D, CHR$(94%) followed by CHR$(68%). The /UP_ARROW qualifier of the SET TERMINAL command sets this characteristic.

255% The system treats control characters as such. The /NOUP_ARROW qualifier of the SET TERMINAL 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.

This byte applies only to interfaces that support DATA/STOP/PARITY features. When you use this byte with these interfaces, it overrides the setting of byte 15. In addition, when you use byte 20, byte 14 must be set to a nonzero value.

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

0% No change.

128% Return the permanent characteristics.

255% Set the characteristics for a terminal to always default to permanent characteristics when the terminal is released (for example, logout or kill). The /PERMANENT qualifier of the SET TERMINAL command determines this value (requires HWCFG privilege).

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

0% No change.

128% The system treats an incoming ESC, CHR$(27%), character as a line terminating character and echoes it as the $ character. The /NOESCAPE_SEQUENCE qualifier of the SET TERMINAL command sets this characteristic.

255% The system treats an incoming ESC, CHR$(27%), character and the following incoming characters as a special escape sequence. See Chapter 4 for a description of incoming escape sequences. The /ESCAPE_SEQUENCE qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

128% Disable (clear) the private delimiter. The /NODELIMITER qualifier of the SET TERMINAL command sets this characteristic.

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. You cannot use the delimiter with INPUT, INPUT LINE, or MAT INPUT statements. These statements recognize only standard delimiters. See Chapter 4, Table 4-3 and the section "Private Delimiters" for a discussion of delimiters.

The /DELIMITER qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

128% The terminal in use does not have the ESC key that generates CHR$(27%). Therefore, translate ALT MODE, CHR$(125%), and PREFIX, CHR$(126%), to CHR$(27%). The /ALT_MODE qualifier of the SET TERMINAL command sets this characteristic.

255% The terminal in use has an ESC key that generates CHR$(27%). Therefore, do not translate CHR$(125%) and CHR$(126%) but treat them as their ASCII characters, right brace (}) and tilde (~). The /NOALT_MODE qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

128% Disable the CTRL/R and CTRL/T facilities. The /NOCONTROL=R AND /NOCONTROL=T qualifiers of the SET TERMINAL command set this characteristic.

255% Enable the CTRL/R and CTRL/T facilities. The /CONTROL=R and /CONTROL=T qualifiers of the SET TERMINAL command set this characteristic.

The CTRL/R facility retypes your terminal's pending input buffer. CTRL/T produces a status report for the current keyboard (unless this byte is set to 128 to disable CTRL/T). This byte is only for compatibility with previous releases. When writing new applications, use Part II of the call instead.

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

0% No change.

128% Define XOFF/XON processing such that the keyboard resumes typeout and echo only after XON or CTRL/C is typed. The /RESUME=CONTROL_C qualifier of the SET TERMINAL command sets this characteristic.

255% Define XOFF/XON processing such that the keyboard resumes typeout and echo when any character is typed after XON. The /RESUME=ANY qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

128% Treat BREAK key as a null. The /NOBREAK qualifier of the SET TERMINAL command sets this characteristic.

255% Translate BREAK key to CTRL/C. The /BREAK qualifier of the SET TERMINAL command sets this characteristic.

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

0% No change.

128% Enable broadcast to the terminal. The /BROADCAST qualifier of the SET TERMINAL command sets this characteristic.

255% Disable broadcast to the terminal. The /NOBROADCAST qualifier of the SET TERMINAL command sets this characteristic.

Data Returned

Bytes	Meaning
3	This byte returns the status of the keyboard specified in byte 4 of the data passed. The following bit tests show the information returned:

Value	Information
Byte 3 AND 1%<>0%	Disabled keyboard or pseudo keyboard that is not in use.
Byte 3 AND 126%=0%	No job owns keyboard.
Byte 3 AND 126%<>0%	(Byte 3 AND 126%)/2 is the job number that owns the keyboard.
Byte 3 AND 128%<>0%	Modem line hung up or pseudo keyboard that is not in use.

5-28 These bytes return values that define the current keyboard characteristics, with three exceptions:

- If you specify 128 in byte 21, then the call returns the permanent terminal characteristics.
- Byte 20 is always returned as 0.

- Byte 19 returns a code that defines the type of interface for the line, where:

Value	Type
0	DL11A, DL11B
2	reserved
4	DL11C, DL11D
6	DL11E
8	pseudo keyboard
10	DJ11
12	DH11
14	DZ11/DZV11
16	DHV11/DHU11

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

16 The 8-bit characteristic is OFF.

24 The 8-bit characteristic is ON.

Privileges Required

None	Set the characteristics of your own terminal
HWCFG	Set the characteristics of a terminal other than your own, or set permanent terminal characteristics

Possible Errors

Meaning	ERR Value
?PROTECTION VIOLATION You do not have sufficient privilege to perform any of these operations:	10

- Read the characteristics of a terminal not opened or assigned to your job.
- Change the characteristics of a terminal other than your job's console terminal (KB:).
- Change the speed setting for your terminal. (Byte 14 or 16 is nonzero.)
- Set the permanent characteristics for a remote terminal line (byte 21 is nonzero).

?ILLEGAL SYS() USAGE 18

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 the calling job is detached.

Discussion

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

If you do not have HWCFG privilege, you can read the characteristics of any terminal that you have opened or assigned but can change terminal characteristics only for your job's console terminal (KB:). In addition, you cannot set speed or permanent characteristics.

Byte 15 sets the output parity bit, data length, and 8-bit characteristic. When the 8-bit characteristic is OFF, incoming data is trimmed to 7 bits (when not in binary mode). When the 8-bit characteristic is ON, incoming data is not trimmed. As a result, all 8 data bits are passed to the user program. DIGITAL recommends you set the 8-bit characteristic to OFF on all 7-bit terminal lines because some terminals send 7-bit data with the eighth bit set rather than cleared. Setting the 8-bit characteristic to OFF also ensures that terminals configured to send 7 data bits with a parity bit, connected to a line configured for no parity checking, will work as they did in the past.

If you want to use the 8-bit feature with parity on a terminal, you must also set up the data length properly. The data length is the number of actual data bits transmitted on the line. The default data length in RSTS/E is 8 bits, and the default parity setting is "disabled."

Normally, when you enable parity, the number of data bits decreases to seven. This would have the wrong effect on 8-bit terminals. Therefore, the system suppresses the changing of the data size when the 8-bit characteristic is set. If you want to suppress the data size change without using the 8-bit setting, set bit 4 in byte 15 when changing the parity setting.

The SET TERMINAL commands use this call to set terminal characteristics. See the *RSTS/E System Manager's Guide* for more information.

Set Terminal Characteristics - Part II

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(16%), the set terminal characteristics code.
3	CHR$(1%), the code to specify part II of the SYS call.
4	CHR$(N%), where N% is 255% for the current keyboard (requires no privilege) or is the keyboard number of the terminal to alter (requires HWCFG privilege).
5	CHR$(N%), where N% is the terminal type code. Legal values (0%-255%) are:

Value	Code
0%	No change
1%	Unknown
2%	LA36
3%	VT52
4%	VT55
5%	LA180S
6%	VT100
7%	LA120
8%	LA12
9%	LA100
10%	LA34
11%	LA38
12%	LA50
13%	VT101
14%	VT102
15%	VT125
16%	VT131
17%	VT132
18%	VT220
19%	VT240
20%	VT241
21%	VT105
22%	VK100
23%	RT02
24%	LA30
25%	VT50
26%	VT50H
27%	VT05
28%	VT05B

29%	LA30S
30%	2741
31%	ASR33
32%	KSR33
33%	ASR35
34%	KSR35
35%-127%	Reserved
128%-255%	Available for customer use

The /TYPE qualifier of the SET TERMINAL command sets this characteristic.

6-20 Reserved; should be 0.

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

Value	Meaning
0%	No change.
128%	Return the terminal's permanent characteristics.
255%	Set the characteristics for a terminal always to default to permanent characteristics when the terminal is released (for example, logout or kill). The /PERMANENT qualifier of the SET TERMINAL command determines this value (requires HWCFG privilege).

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

Value	Meaning
0%	No change.
6%-255%	The terminal's new input buffer quota. The default value is 6. The /BUFFER_QUOTA qualifier of the SET TERMINAL command sets this characteristic (requires HWCFG privilege). See Discussion

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

Value	Meaning
0%	No change.
1%	Enable the CTRL/C control character.
2%	Enable the CTRL/T control character.
4%	Enable the CTRL/R control character.
8%	Enable the CTRL/X control character.
16%	Enable the autobaud facility.

CTRL/C usually halts execution of the current command or program and returns control to the job keyboard monitor. CTRL/T produces a status report for the current keyboard. CTRL/R retypes your terminal's pending input buffer. CTRL/X deletes the current line as well as all type-ahead. The /CONTROL qualifier of the SET TERMINAL command sets the control characteristics.

The /AUTOBAUD qualifier of the SET TERMINAL command sets the autobaud characteristic. See the Discussion for more information on the autobaud facility.

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

Value	Meaning
0%	No change.
1%	Disable the CTRL/C control character.
2%	Disable the CTRL/T control character.
4%	Disable the CTRL/R control character.
8%	Disable the CTRL/X control character.
16%	Disable the autobaud facility.

The /NOCONTROL qualifier of the SET TERMINAL command clears the control characteristics.

The /NOAUTOBAUD qualifier of the SET TERMINAL command clears the autobaud characteristic.

25-26 CHR$(N%)+CHR$(SWAP%(N%)), where N% is:

Value	Meaning
0%	No change.
1%	Set ANSI escape sequences (/ANSI).
2%	Set advanced video (/ADVANCED_VIDEO).
4%	Set 132 columns (/132_COLUMNS).
8%	Set printer port (/PRINTER_PORT).
16%	Set ReGIS graphics (/REGIS).
32%	Set sixel graphics (/SIXEL).
64%	Set Katakana character set (/KATAKANA).
128%	Set selectively eraseable characters (/SELECT_ERASE).
256%	Set dynamically redefinable character sets (/LOADABLE).
512%	Set user defined keys, UDKs (/USER_DEFINED_KEYS).
1024%	Set local copy (/LOCAL_ECHO).
2048%	Set noninteractive mode. See Discussion.

These terminal capability flags let a program check which functions a terminal can perform. If you set these flags,

no special processing is performed. See the documentation on your particular terminal for more information on its capabilities.

The qualifier of the SET TERMINAL command that sets a characteristic is shown in parenthesis after each bit value description.

27-28 CHR$(N%)+CHR$(SWAP%(N%)), where N% is:

Value	Meaning
0%	No change.
1%	Clear ANSI escape sequences.
2%	Clear advanced video.
4%	Clear 132 columns.
8%	Clear printer port.
16%	Clear ReGIS graphics.
32%	Clear sixel graphics.
64%	Clear Katakana character set.
128%	Clear selectively eraseable characters.
256%	Clear dynamically redefinable character sets.
512%	Clear user defined keys (UDKs).
1024%	Clear local copy.
2048%	Clear noninteractive mode. See Discussion.

29-30 Reserved; should be 0.

The qualifier of the SET TERMINAL command that clears a characteristic is the same one shown in parenthesis after each bit value description in bytes 25-26, with the NO prefix appended.

Data Returned

Bytes	Meaning
5	This byte returns the value of the current terminal type.
23	This byte returns the current control character flag status; that is, those flags that are set.
25-26	These bytes return the current capability flag status; that is, those flags that are set.

Privileges Required

None	Set the characteristics of your own terminal
HWCFG	Set the characteristics of a terminal other than your own, or set permanent terminal characteristics

Possible Errors

Meaning	ERR Value
?PROTECTION VIOLATION You do not have sufficient privilege to perform any of these operations: o Read the characteristics of a terminal not opened or assigned to your job. o Change the characteristics of a terminal other than your job's console terminal (KB:). o Change the terminal's buffer quota.	10
?ILLEGAL SYS() USAGE One of the following occurred: o The keyboard number specified in byte 4 of the call is out of the range of valid keyboard numbers. o The current keyboard is specified (byte 4=255) but the calling job is detached. o The value in byte 22 is not within the range of 6-255.	18

Discussion

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

If you do not have HWCFG privilege, you can read the characteristics of any terminal that you have opened or assigned but can change terminal characteristics only for your job's console terminal (KB:). In addition, you cannot set speed or permanent characteristics.

Set Terminal Characteristics - Part II
F0=16

The SET TERMINAL command uses this call to set terminal characteristics. See the *RSTS/E System Manager's Guide* for more information on SET TERMINAL.

Byte 22 sets the input buffer quota. The default quota value is 6. Since there are 30 characters in a buffer, this means that terminal service attempts to buffer 180 (6 times 30) characters before sending the device an XOFF. Note that there is no guarantee that a terminal can allocate its full buffer quota, because a heavy system load may leave less than the terminal's full buffer quota available. Also, excessive use of this feature to allocate large numbers of buffers to several terminals can create a shortage of small buffers.

Bit value 16 in byte 23 sets the autobaud characteristic. Autobaud enables RSTS/E to detect and then set terminal speed on a particular multiplexed line. Once the characteristic is set, the user types carriage return (CR) until the system prompts with 'User:'. You can set the autobaud feature on DZ11, DZV11, DZQ11, DH11, DHV11 and DHU11 multiplexers. Autobaud supported speeds are 110, 150, 300, 600, 1200, 1800, 2400, 4800 and 9600 baud. RSTS/E does not support split speeds when using the autobaud feature. Bit value 16 in byte 24 clears the autobaud characteristic. The autobaud characteristic is meaningful only when setting permanent characteristics.

Bit value 2048% in bytes 25-26 sets noninteractive mode. You can use the noninteractive characteristic with devices such as printers that are attached to terminal lines. These devices do not function as interactive terminals; in particular, they are never used to initiate a terminal session. If the noninteractive characteristic is set, the system ignores any characters received from the terminal if the terminal is not owned by a job. If the terminal is owned by a job, the system processes characters normally. Bit value 2048% in bytes 27-28 clears noninteractive mode. The noninteractive characteristic is meaningful only when setting permanent characteristics.

See the section "Directory Lookup" for a description of this call.

Enable and Disable Disk Caching

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(19%), the enable and disable caching code.
3	CHR$(N%), where N% is:

Value	Meaning
0%	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.
1%	To disable all caching.
128%	Return the current caching parameters. A 128 value in this byte does not enable or disable data 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$(SWAP%(L%)), where L% sets a limit on the total number of cache clusters that can be used. If L% is 0, the current limit is used. See Discussion. If L% is nonzero, it specifies an upper limit on the number of clusters in the cache. Note that if the amount of XBUF available to the cache is less than L%, the cache does not exceed XBUF.
7-8	CHR$(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 nonzero, 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 nonzero, 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 Chapter 1).

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 Allow use of the small buffer pool.

M%=128 Do not use the small buffer pool.

13 CHR$(T%), where T% is the new value of the cache replacement time. Specify 0% for no change.

14-30 Reserved; should be 0.

Data Returned

Bytes	Meaning
1-2	Internal coding.
3	Current cache setting and available options:

Value	Meaning
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.
4-13	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 installation.	
14-30	Not used.	

Privileges Required

TUNE

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 You do not have the TUNE privilege.	10
?MISSING SPECIAL FEATURE Caching was not enabled for the system during system installation.	66

Discussion

Bytes 1, 2, and 3 of this call enable or disable the FIP buffering module that controls directory caching. The SET SYSTEM command uses these bytes.

If the system manager installed user data caching on the system during system installation, bytes 1-13 enable or disable user data caching and set the parameters of the cache. The system manager defines the total size of XBUF during system installation, 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. For example, if the system manager defines a 40K word XBUF at system installation, the SYS call can define the directory and data portions of the cache as 25K words each. That is, data can use the space in the cache up to a maximum of 25K words, which leaves a minimum of 15K words 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 both the cache and DECnet/E use XBUF, limiting the cache guarantees that space is always available in XBUF for DECnet/E. Note that the system frees the amount of memory allocated to the cache for other use when it is not performing caching.

Byte 4 of the call sets the cache cluster size. This parameter controls the number of contiguous blocks that are copied from the disk to the cache whenever a file or directory is cached (see Chapter 1). 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 make sure 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 is allowed (see Chapter 1). See 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 start-up. The default settings are a cache cluster size of 4, with no limits on directory, data, or total cache size, and a cache replacement value of 30. The system manager can reset these defaults with an INIT option, as the *RSTS/E System Installation and Update Guide* describes.

Date and Time Conversion

Data Passed

Bytes	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:

Value	Meaning
D%=0	Use the system default format.
D%<0	Use alphabetic date format.
D%>0	Use ISO numeric date format.

7-16	Reserved; should be 0.
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:

Value	Meaning
T%=0	Use the system default format.
T%<0	Use AM/PM time format.
T%>0	Use 24-hour time format.

21-30	Reserved; should be 0.

Data Returned

Bytes	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 nulls.

17-20 Same as data passed.

21-30 The time string, padded to the right with nulls.

Privileges Required

None.

Possible Errors

No errors are possible; however, if bytes 3-4 or 17-18 contain illegal date or time values, unpredictable output may be generated.

Discussion

Use this call in programs that need to override the system date and time defaults.

System Logical Names

RSTS/E allows users to access devices by logical names as well as by physical names. Logical names that apply to all users are called system logical names. On all systems, users can refer to a disk by its pack identification or a logical name that replaces the pack identification.

This SYS call allows you to add, remove, change, and list system logical names. The total number of additional system logical names allowed is limited by the size of the extended buffer pool (XBUF).

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 PPN associated with the logical name.

The second part of the logical name table is optional. Space for the table is taken from XBUF. 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. In addition, an entry in the second part of the table can have a PPN 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 you associate the system logical name SCRATCH with account [100,100] on RP04 unit 2, and open the file SCRATCH:[200,240]OTHER.DAT, the system attempts to access the file OTHER.DAT on RP04 unit 2 under account [200,240]. The specification SCRATCH:OTHER.DAT refers to the file OTHER.DAT in account [100,100] on RP04 unit 2, the account associated with the logical name SCRATCH.

The Mount and Dismount SYS calls (SYS 3) 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 disk being mounted (unless NOLOGICAL is specified or the logical name is already in use). The Dismount call removes a pack identification or logical name from the entry for that disk.

The System Logical Name call can change or remove a name (or pack identification) in the first part of the table. The Logical Name SYS call can add or remove entries in the second part of the table.

This call has four subfunctions:

- Add New Logical Names
- Remove Logical Names
- Change Disk Logical Name
- List Logical Names

The following sections describe the variations of the Logical Name SYS call.

Add New Logical Names
(F0=21) (UU.SLN)

Add New Logical Names

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(21%), the system logical name code.
3	CHR$(1%), to add a new entry in the logical name table.
4	Reserved; should be 0.
5-6+	PPN to be associated with this logical name. If these bytes are 0, no account is associated with the logical name.
7-12	The system logical name, in Radix-50 format.
13	CHR$(N%), where N% is one of the following values:

Value	Meaning
0%	Do not replace an existing logical name. See Discussion.
1%	Replace an existing logical name with the new information. See Discussion.

14-22	Reserved; should be 0.
23-26+	The device name and unit number to which the logical name applies.
27-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
13	If you passed a value of 1% in byte 13, the call returns:

Value	Meaning
0	The logical name did not already exist
1	The logical name existed and was replaced.

Privileges Required

INSTAL

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME No name is found in bytes 7 through 12, or the name found contains nonalphanumeric characters.	2
?ACCOUNT OR DEVICE IN USE The name specified in bytes 7 through 12 duplicates one already in either the first or second part of the table.	3
?NOT A VALID DEVICE The device specification in bytes 23 through 26 is illegal or the related device is not configured on the system.	6
?ILLEGAL SYS( ) USAGE You specified an illegal value in byte 3.	18
?NO BUFFER SPACE AVAILABLE There is no more room on the system for a new entry. To free up an entry, issue the remove logical name SYS call.	32

Discussion

System logical names can be up to nine characters long. When adding a name of fewer than nine characters, you must fill the extra space at the end in bytes 7-12 with zeros (RAD50 blanks). This call scans the entire system logical name table for the name given in bytes 7 through 12.

Byte 13 specifies whether to replace an existing logical name. If you specify a value of 1% (replace), the system deassigns the existing logical name. The system follows these procedures to determine whether to add the new logical name to the first or second part of logical name table:

1. If the device name specified in bytes 23-26 is not a disk, the system adds the logical name to the second part

2. If the device name is a disk and you specified a PPN in bytes 5-6, the system adds the logical name to the second part

3. If the device name is a disk and you did not specify a PPN in bytes 5-6, the system checks to see if the disk is mounted. If the disk is not mounted, the system adds the logical name to the second part. If the disk is mounted, the system follows these procedures:

    1. If a logical name for the disk does not exist in the first part of the table, the system adds the new logical name to the first part

    2. If a logical name for the disk exists in the first part of the table, the system adds the new logical name to the second part.

The DCL ASSIGN/SYSTEM command uses this call.

Remove Logical Names

Data Passed

Bytes	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	Reserved; should be 0.
7-12	The system logical name, in Radix-50 format.
13-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME No name is found in bytes 7 through 12, or the name found contains nonalphanumeric characters.	2
?CAN'T FIND FILE OR ACCOUNT The name specified in bytes 7 through 12 is not currently defined as a logical name.	5
?ILLEGAL SYS() USAGE You specified an illegal value in byte 3.	18

Remove Logical Names
(F0=21)

Discussion

This call scans the entire system logical name table for the name specified in bytes 7 through 12. 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.

When you remove a system logical name of fewer than nine characters, you must fill the extra space at the end in bytes 7-12 with zeros (RAD50 blanks).

Change Disk Logical Name

Data Passed

Bytes	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	Reserved; should be 0.
7-12	The system logical name, in Radix-50 format.
13-22	Reserved; should be 0.
23-26+	The name and unit number of the disk device whose logical name is to be changed.
27-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME No name is found in bytes 7 through 12, or the name found contains nonalphanumeric characters.	2
?ACCOUNT OR DEVICE IN USE The logical name specified in bytes 7 through 12 duplicates one already in either the first or second part of the table.	3

?CAN'T FIND FILE OR ACCOUNT 5
The disk specified in bytes 23 through 26 is not configured on this system.

?NOT A VALID DEVICE 6
The device specified in bytes 23 through 26 is illegally formatted or is not a disk.

?ILLEGAL SYS() USAGE 18
You specified an illegal value in byte 3.

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 12 is placed in the entry. This call accepts system logical names up to nine characters long. When changing a logical name of less than nine characters, you must fill the extra space in bytes 7-12 with zeros (RAD50 blanks).

List Logical Names

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(21%), the system logical name code.
3	CHR$(2%), to list the entries in the logical name table.
4	Reserved; should be 0.
5-6	CHR$(N%), where N% is the index of the logical name entry to be listed.
7-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1-4	Not used.
5-6	PPN of the account associated with the logical name. 0% if no PPN is associated with the logical name.
7-12	The system logical name, in Radix-50 format.
13-22	Not used.
23-26+	The device name and unit number of the Nth logical. 0% if no device name is associated with the logical name.
27-30	Not used.

Privileges Required

None.

List Logical Names
(F0=21)

Possible Errors

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT The index entry specified in bytes 5-6 is out of range.	5
?ILLEGAL SYS() USAGE You specified an illegal value in byte 3.	18

Discussion

This call scans the system logical name table and lists the logical name that corresponds to the index number passed in bytes 5-6. You can list all logicals by repeated calls with an index value, starting at 0 and increasing the index value by 1 each time.

System Call for Local Interjob Communication (Send/Receive)

See Chapter 8 for a description of the Send/Receive system function call. See Chapter 9 for a description of the System Call for Print/Batch Services (PBS), a subfunction of the Send/Receive call.

Add, Remove, and List System Files

Swap files on RSTS/E are dynamically added at the start of timesharing and can be added and removed during timesharing. Swap files must be removed to properly shut down timesharing. Optionally, you can add three other system files during timesharing to optimize system performance: the overlay file, the error message file, and the DECnet/E system file.

This SYS call adds and removes these system files and also lets you obtain their file specifications. Through the INIT.SYS program and the DCL INSTALL command, 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. See the *RSTS/E System Manager's Guide* and the *RSTS/E System Installation and Update Guide* for details on these operations. Refer to the *DECnet/E System Manager's Guide* for more information on the DECnet/E system file, also called the DECnet/E volatile parameter file.

The following sections describe the three subfunctions of this SYS call:

- Add System Files
- Remove System Files
- List System Files

Add System Files

Data Passed

Bytes	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 add:

Value	Meaning
0%	Swap file 0
1%	Swap file 1
2%	Illegal - generates error
3%	Swap file 3
4%	Overlay file
5%	Error message file
6%	DECnet/E system file

To specify codes 0-5, you need read/write access to the file. Code 6 requires only read access to the file.

4	CHR$(1%), to add a system file.
5-6	Reserved; should be 0.
7-10+	To add a file that currently exists in directory [0,1] and has a file type of .SYS, specify 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 makes sure that it exists, is large enough, and has proper characteristics.
11-22	Reserved; should be 0.
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	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL Add system files

WRTNFS Add a non-file-structured disk as a swapping device

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME No name is specified in bytes 7 through 10 when an overlay, error message, or DECnet/E system file is being added; or the name specified contains nonalphanumeric characters.	2
?ACCOUNT OR DEVICE IN USE A swap 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 128 blocks and the error file at least 16 blocks.) If a swap file is being added to a file-structured device, this error means that the file is not long enough to store even one 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 file type does not exist in directory [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 swap 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	10

the file specified in bytes 7 through 10 is bad (that is, it is not contiguous or is currently open).

?NAME OR ACCOUNT NOW EXISTS 16
The system file being added as described in byte 3 is already installed on the system.

?ILLEGAL SYS() USAGE 18
The number specified in byte 3 is either 2 or is greater than 6. The swap file for file 2 must exist on the system disk and cannot be added during timesharing. System files to be added are defined only by the values 0, 1, 3, 4, 5, and 6.

?DISK PACK IS NOT MOUNTED 21
A system file is being added to a file-structured disk but that disk is not currently mounted. Use the MOUNT command to logically mount the disk before the file is added.

?DEVICE NOT FILE STRUCTURED 30
The device specified in bytes 23 through 26 is not a disk device.

Discussion

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

The *RSTS/E System Installation and Update Guide* 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 resources properly. For example, swap 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 non-file-structured disk is to be added as a swap file, the system manager must make sure that the device is available for such use.

Although this call adds swap file space, it does not alter the job maximum allowed on the system. Adding a swap file merely increases the capability of the system to handle a larger number of jobs. To increase the job maximum after a swap file is added, you must use the Enable Logins SYS call (SYS -1) or the SET SYSTEM/LOGINS command.

Remove System Files

Data Passed

Bytes	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:

Value	Meaning
0%	Swap file 0
1%	Swap file 1
2%	Illegal - generates error
3%	Swap file 3
4%	Overlay file
5%	Error message file
6%	DECnet/E system file

4	CHR$(0%), to remove a system file.
5-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

INSTAL

Possible Errors

	Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE	The swap file to be removed can be properly removed but currently contains one or more swapped-out jobs. The system locks the file and begins 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 10
A swap file is to be removed, but its removal will decrease the swap file space below the limit required to store the maximum number of jobs on the system. To remove the swap 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. An attempt to remove the DECnet/E system file when DECnet/E is still on also causes this error.

?ILLEGAL SYS() USAGE 18
The number specified in byte 3 is either 2 or is greater than 6. The swap file for file 2 must exist on the system disk and cannot be removed during timesharing. System files to be removed are defined only by the values 0, 1, 3, 4, 5, and 6.

Discussion

This SYS call removes a system file from operation. Previously added system files must be removed in order to shut down time-sharing operations. Removing system files for other purposes allows a system manager to adjust system operation without ending timesharing. For example, if a disk currently operating as a swapping device malfunctions during timesharing, the system manager can:

- Decrease the allowed number of logins appropriately
- Remove the swap file for that device
- Dynamically add another device or file to replace the disk as a swap file
- Increase the new allowed number of logins to take advantage of the added swapping space

After shutting down the system, the system manager can disable the malfunctioning unit to allow maintenance and to isolate the device from time-sharing access. Normal time-sharing operations can proceed without further changes to the system.

List System File

Data Passed

Bytes	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 list as follows:

Value	Meaning
0%	Swap file 0
1%	Swap file 1
2%	Swap file 2
3%	Swap file 3
4%	Overlay file
5%	Error message file
6%	DECnet/E system file

4	CHR$(-1%) to list a system file.
5-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
5-6+	PPN. These bytes are 0 if the device is non-file-structured.
7-10+	File name in Radix-50 format. These bytes are 0 if the device is non-file-structured.
11-12+	File type in Radix-50 format. These bytes are 0 if the device is non-file-structured.
13-22	Not used.
23-24+	Device name in ASCII format.
25+	Unit number.
26+	Unit number flag.
27-30	Not used.

Privileges Required

None.

Possible Errors

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT The number specified in byte 3 refers to a file that is not currently installed.	5
?ILLEGAL SYS( ) USAGE The number specified in byte 3 is less than 0 or greater than 6.	18

Discussion

This SYS call returns the file specification of a currently installed system file. When the file is on a non-file-structured device, the call returns the device name. When the file is on a file-structured device, the call returns the device name, PPN, file name, and file type.

Create a Job

The Create a Job SYS call allows appropriately privileged users to create logged-out jobs. In addition, it allows both privileged and nonprivileged users to create jobs that are automatically logged in to an account either to run a program, or to enter a keyboard monitor.

This section presents the data passed and returned for the three functions of this call:

- Create Logged-Out Job
- Create Logged-In Job to Run a Program
- Create Logged-In Job to Enter a Keyboard Monitor

The discussion at the end of this section provides further details on all three functions.

Data Passed -- Logged-Out Job

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(24%), the create a job code.
3	CHR$(N%), where N% is 128% to create the job even if logins are disabled. Specify 0% for N% to create the job only if logins are enabled.
4	Reserved; should be 0.
5-6+	The project-programmer number of the program to run.
7-10+	File name of the program to run, in Radix-50 format.
11-12+	File type of the program to run, in Radix-50 format.
13-22	Ten bytes of information to be 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 run.

27-28 The parameter word to be passed to the program to 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 Reserved; should be 0.

Data Returned

Bytes	Meaning
3	The job number times 2 of the job just created.

Data Passed -- Logged-In Job to Run a Program

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(24%), the create a job code.
3	CHR$(N%), where N% can contain the following bit values when you create a logged-in job to run a program:

Value	Meaning
2%	Do not pass the user logicals of the creating account to the new job. (Contents of core common are still passed.)
4%	Create the job with current privileges equal to "none."
8%	Create the job with the authorized privileges of the new job equal to the authorized privileges of the job's account ANDed with the current privileges of the calling job. If this bit is not set, the new job is created with the authorized privileges of the job's account.
32%	Create the job logged in to the account specified in bytes 13 and 14 (requires GACNT or WACNT privilege).
64%	Create logged in job. You must specify this value for a logged-in job. The new job runs under the caller's account unless you also specify 32%.

	128% Create the job even if logins are disabled. This value is ignored if you do not have JOBCTL privilege. When you do not include this value, the job is created only if logins are enabled.
4	Keyboard to attach the new job to. To indicate KB0: use 128%. Specify a value of 0 in this byte to create a detached job.
5-6+	The PPN of the program to run.
7-10+	File name of the program to run, in Radix-50 format.
11-12+	File type of the program to run, in Radix-50 format.
13-14	The account under which the job will run (requires GACNT or WACNT privilege; bit 32% must be set in byte 3). If you do not have the GACNT or WACNT privilege, set both bytes to 0.
15	Priority of the new job (requires TUNE privilege). If you specify 0, the system uses the caller's values. Use 255% to explicitly specify priority 0. Users without TUNE privilege must set this byte to 0. Note that if you create the job on a pseudo keyboard, the system reduces the priority to the controlling job's priority.
16	Run burst of the new job (requires TUNE privilege). If you specify 0, the system uses the caller's values. Users without TUNE privilege must set this byte to 0.
17	Maximum job size of the new job (requires TUNE privilege). If you specify 0, the system uses the caller's values. Users without TUNE privilege must set this byte to 0.
18-22	Reserved; should be 0.
23-26+	Device containing the program to be run.
27-28	The parameter word.
29-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
3	The job number times 2 of the job just created.

Data Passed -- Logged-In Job to Enter a Keyboard Monitor

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(24%), the create a job code.
3	CHR$(N%), where N% can contain the following bit values when you create a logged-in job to enter a keyboard monitor:

Value	Meaning
2%	Do not pass the user logicals of the creating account to the new job. (Contents of core common are still passed.)
4%	Create the job with current privileges equal to "none."
8%	Create the job with the authorized privileges of the new job equal to the authorized privileges of the job's account ANDed with the current privileges of the calling job. If this bit is not set, the new job is created with the authorized privileges of the job's account.
16%	Enter keyboard monitor instead of running a program. (You must include this value.)
32%	Create job to run under the account specified in bytes 13 and 14 (requires GACNT or WACNT privilege).
64%	Create logged-in job. (You must include this value.) The new job runs under the caller's account unless you also specify 32%.
128%	Create job even if logins are disabled. This value is ignored if you do not have JOBCTL privilege. When you do not include this value, the job is created only if logins are enabled.

4	Keyboard to attach to. The value you specify must be nonzero. To indicate KB0: use 128%.
5-6	Reserved; should be 0.
7-10	Run-time system name in Radix-50 format. The run-time system you specify must be installed and must be a keyboard monitor. Specify 0 for the system's default keyboard monitor, DCL.

11-12	Reserved; should be 0.
13-14	Account under which job will run (requires GACNT or WACNT privilege; the value 32% must be included in byte 3). If you do not have the GACNT or WACNT privilege, specify 0 in both bytes.
15	Priority of new job (requires TUNE privilege). If you specify 0, the system uses the caller's values. Use 255% to explicitly specify priority 0. Users without TUNE privilege must set this byte to 0. Note that if you create the job on a pseudo keyboard, the system reduces the priority to the controlling job's priority.
16	Run burst of the new job (requires TUNE privilege). If you specify 0, the system uses the caller's values. Users without TUNE privilege must set this byte to 0.
17	Maximum job size of new job (requires TUNE privilege). If you specify 0, the system uses the caller's values. Users without TUNE privilege must set this byte to 0.
18-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
3	The job number times 2 of the job just created.

Privileges Required

None	Create a job in your own account
GACNT	Create a job in another account in the group
WACNT	Create a job in any account, or create a logged-out job
EXQTA	Ignore detached-job and total-job quota on create
JOBCTL	Create a job even if no logins are set
TUNE	Specify a priority, run-burst, or maximum job size when creating a job

Possible Errors

Meaning	ERR Value
?NO ROOM FOR USER ON DEVICE The new job cannot be created. Probable causes are: o Further logins are disabled, and byte 3 does not include the value 128%. Or, logins are disabled and byte 3 contains the value 128%, but you do not have JOBCTL privilege. o The system's job or swap slots are currently full.	4
?CAN'T FIND FILE OR ACCOUNT You have sufficient privilege to create a logged-in job, but the system cannot log the job in. Possible causes are that you specified a nonexistent account or a no-user account.	5
?NOT A VALID DEVICE The keyboard number specified in byte 4 is invalid.	6
?DEVICE NOT AVAILABLE The keyboard specified in byte 4 is open, is in use, or is not assigned to the calling job. A user without sufficient privilege can also get this error if the system manager has restricted the device to users with the DEVICE privilege.	8
?PROTECTION VIOLATION You do not have the necessary privilege to perform the specified operation.	10
?ILLEGAL SYS() USAGE You are trying to create a logged-in job to enter a keyboard monitor, but did not supply a keyboard number in byte 4.	18
?NO BUFFER SPACE AVAILABLE You are trying to create a logged-in job, but not enough XBUF is available for temporary storage of your current job's core common and user logicals.	32

Note that when you create a logged-in job, you can also get any error that can occur while logging in, running a program, or those errors associated with the run-time system. If such an error occurs, the monitor kills the new job and returns the error to your job instead.

Discussion

Creating a Logged-Out Job

If you have the WACNT privilege, you can use this call to create logged-out jobs. To do so, specify the complete file specification of the program the created job is to run in bytes 7-12. You must include the PPN in bytes 5 and 6; there is no default.

The program must be compiled (executable).

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

Because the created job runs logged out, the system kills the job if it fails or exits.

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

Note that this option is provided for compatibility with previous versions of RSTS/E. Due to the limitations of the Create Logged-Out Job function, new applications should use the Create Logged-In Job function instead.

Creating a Logged-In Job

Both appropriately privileged and nonprivileged users can create jobs that are automatically logged in to an account, either to run an executable program or to enter a keyboard monitor at the P.NEW entry point (see the *RSTS/E System Directives Manual*).

If you specify 64% in byte 3, the job is created and logged in to your own account. If you have GACNT or WACNT privilege (as appropriate), you can create the job to run under an account other than your own account by specifying 64% and 32% in byte 3 (specify the account under which the job will run in bytes 13 and 14). In addition, users who have the JOBCTL privilege can create the job even if logins are disabled by adding 128% in byte 3.

The value in byte 4 determines if the new job is logged in detached or attached. If this byte is 0, the new job is created logged-in and detached. If the value in this byte is nonzero, the system sets the sign bit of the byte to 0, and uses the resulting value as the keyboard number to attach to. The terminal you specify in this byte must be a free terminal or must be allocated to the calling job. In addition, if you do not have the DEVICE privilege you cannot specify a terminal that is restricted.

When a new job is created logged-in, it receives a copy of all of the core common and user logicals of the calling job. You can set bit 1 in byte 3 to suppress passing of user logicals. XBUF must be available because it is used for temporary storage of this data.

If the new job is detached, its channel 0 Device Data Block (DDB) points to the console terminal of the calling job.

If you have the TUNE privilege, you can pass priority, run burst, and memory maximum values to the new job in bytes 15, 16, and 17. If you pass zeros in these bytes, the system takes them from the calling job. If you do not have TUNE privilege, the system automatically passes the values from the calling job to the new job, so specify 0 in all three bytes. Note that if you create the job on a pseudo keyboard, the system reduces the priority to the controlling job's priority if you specify a higher priority.

Creating a Logged-In Job to Enter a Keyboard Monitor

As previously mentioned, both appropriately privileged and nonprivileged users can use this call to enter a keyboard monitor instead of running a program.

To enter a keyboard monitor, you must specify 64% and 16% in byte 3. See the Data Passed section for other values you can specify.

Specify the name of the keyboard monitor to enter in bytes 7-10. When you specify a keyboard monitor, the new job must be attached to a keyboard (indicated in byte 4). If you specify zeros in bytes 7-10, the system uses the system's default keyboard monitor (DCL). Note that the keyboard monitor you specify must be an installed run-time system.

Bytes 13-17 are used as previously described for logged-in jobs that run a program.

Wildcard PPN Lookup

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(25%), the look up an account number by index code.
3-4	CHR$(N%)+CHR$(SWAP%(N%)), where N% is the index of the requested PPN. If N% is 0, the call returns the PPN of the first account on the disk that matches the wildcard specification. If N% is nonzero, the call returns the PPN of the N+1 account on the specified disk that matches the wildcard specification.
5-6	The requested PPN. A value of 255 in either field represents a wildcard. A value 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 bytes 3-4 and 5-6 are zero, the user's account is looked up. If bytes 3-4 and 5-6 are nonzero and contain a PPN with no wildcard characters, the call returns error 5.
7-22	Reserved; should be 0.
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.
27-30	Reserved; should be zero.

Data Returned

Bytes	Meaning
3-4	Internal code.
5-6+	The PPN located by the call.
7-30	Not used.

Privileges Required

DEVICE	Access a restricted device

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 (also see bytes 5-6).	5
?DEVICE IS RESTRICTED The disk is restricted. You need DEVICE privilege 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 PPN. The wildcard account specification is in the form that the File Name String Scan call (SYS -10, -23) returns.

Return Job Status

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(26%), the return job status code.
3	CHR$(J%), where J% is the number of the job for which status is desired. If J% is 0%, information on the caller's job is returned. If the caller does not have JOBCTL privilege, J% is forced to 0%. Note that if a job is running on a pseudo keyboard, its controlling job may examine the controlled job regardless of privilege.
4	CHR$(S%), where S% is 0%, 1%, or 2%. The value of S% determines the information returned on the job (see Data Returned).
5-30	Reserved; should be 0.

Data Returned

If S% is 0%:

Bytes	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.
4	Keyboard number of the job's console, if the job is attached. The job is detached if (M%(4%) AND 128%)<>0%; in this case, the keyboard number is equal to NOT M%(4%). A program that uses this byte should test both possibilities even if the program is not designed to run detached, because any program can become detached if a dial-up line loses the telephone connection, or if a sufficiently privileged user detaches the job.
5	If the job is attached to a pseudo keyboard, this byte contains the controlling job's job number times two, plus one; otherwise, it is 0.

6	If the job is swapped out, this byte contains the job's swap slot location; otherwise, it 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 PPN.
23-26	The name of the job keyboard monitor 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%:

Bytes	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.
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. 0 if the caller does not have TUNE privilege.
18	The job's allotted run burst in tenths of a second. 0 if the caller does not have TUNE privilege.
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 nonkeyboard 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 address of the job's first receiver identification block; otherwise, these bytes are 0.

If S% is 2%:

Bytes	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.
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	Job's current I-space size.
6	Job's current D-space size.
7-14	Job's current privilege mask.
15	Job's access type:

Value	Type
0%	Local
1%	Dialup
2%	Batch
4%	Network
6%	Server (jobs started by DECnet)

Other values reserved

16-30	Reserved.

Privileges Required

None	Read the status of your own job or a job controlled by your job
JOBCTL	Read the status of any job
TUNE	Read the priority or run burst of any job

Return Job Status
F0=26

Possible Errors

Meaning	ERR Value
?PROTECTION VIOLATION The job whose status is requested does not exist.	10
?ILLEGAL SYS( ) USAGE The job number whose status is requested is less than zero or greater than JOB MAX.	18

Set, Clear, or Read Current Privileges

This call has two subfunctions:

- Set/Clear Current Privileges
- Read Current Privileges

Set/Clear/Read Current Privileges

To set or clear current privileges, specify the following bytes:

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(28%), the set/clear/read privileges code.
3-10	The privilege mask of bits to set. Each set bit corresponds to a bit that is to be turned ON in the privilege mask.
11-14	Reserved; should be 0.
15-22	The privilege mask of bits to clear. Each set bit corresponds to a bit that is to be turned OFF in the privilege mask.
23-30	Reserved; should be 0.

To read current privileges, specify the following bytes:

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(28%), the set/clear/read privileges code.
3-30	Reserved; should be 0.

Set/Clear/Read Current Privileges
F0=28

Data Returned

Bytes	Meaning
1-2	Not used.
3-10	Privileges now in effect.
11-30	Not used.

Privileges Required

None.

Possible Errors

None.

Discussion

This SYS call reads the current privilege mask or selectively sets and/or clears bits in it. This call is distinct from SYS call -21, Drop/Regain Temporary Privileges, which only applies when you are running a privileged program and then update the entire current mask at once.

You pass the bits to set in bytes 3-10. You pass the bits to clear in bytes 15-22. Before setting the bits passed in bytes 3-10, the system ANDs the bits passed with the authorized mask to prevent the caller from setting unauthorized bits. Note that this happens whether or not temporary privileges are in effect. In other words, a program with temporary privileges can use this SYS call to drop any single privilege, but it can regain only those that the user is authorized to have.

The privileges in effect on completion of the SYS call are returned in bytes 3-10. To simply read the current privileges, issue the SYS call with zeros in bytes 3-30.

If a privileged program wants to find out what privileges the user (as opposed to the program) has, it must perform a Drop Temporary Privileges SYS call (SYS -21) followed by a Read Privileges SYS call. This makes sure that it will read the normal mask.

See Chapter 1 for more information about privileges.

Stall/Unstall System

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(29%), stall/unstall system code.
3	CHR$(N%), where N% is:

Value	Meaning
0%	Return the system to normal ("unstalled") state.
1%	Stall the system (suspend all active jobs).

4-30	Reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

HWCTL

Possible Errors

Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE You issued a stall request and the system was already stalled.	3
?ILLEGAL SYS() USAGE You issued an unstall request and the system was not stalled.	18

Stall/Unstall System
F0=29

Discussion

This call suspends all currently active jobs except for the calling job. The purpose of the call is to stop all system activity before spinning down the system disk to change the removable platter for other applications. This call is used primarily for the RC25 disk in configurations where the system disk is on the fixed part of the RC25 and you want to exchange the removable part of that same drive. The DCL SET SYSTEM/HOLD and SET SYSTEM/RELEASE commands use this call.

The calling job is not affected when the system is stalled and can continue to perform any operation. However, if the system is stalled for the purpose of spinning down the system disk, an attempt to do disk I/O to that disk results in an I/O error.

Note that logins are disabled while the system is stalled.

Third Party Privilege Check

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(31%), the third-party privilege check code.
3-4	Reserved; should be 0.
5-6+	PPN to enable the third-party privilege check. Zero to disable the third-party privilege check.
7-14	A privilege mask.
15-30	Reserved; should be 0.

Data Returned

None.

Privileges Required

None.

Possible Errors

Meaning	ERR Value
?NO BUFFER SPACE AVAILABLE A small buffer is needed to store the third party information, but none is available.	32

Discussion

This call enables or disables third-party privilege checking. This call is primarily for server programs such as Print/Batch Services (PBS) that run detached under a highly privileged account and perform functions on behalf of normally less privileged users.

To maintain system security, these server programs must enforce appropriate privilege restrictions that apply to the user issuing requests to the server. For example, a print server must enforce the file access restrictions on the user requesting a printout, but does

not want to enforce the access restrictions to the printer device. Third party privilege checking allows the server program to correctly perform these checks.

In this call, the server job specifies the PPN and privileges of the requesting user. From that point, the monitor performs every privilege check twice: once against the privileges of the server job itself, and once against the privileges of the third party. This continues until the server job issues the call again to cancel third party privilege checking.

For example, a print server issues the third party privilege check call to enable third party privilege checking prior to opening the file to be printed. If the requester is not allowed to access the file, the open fails in the usual manner. The account number of the requester is part of the information passed in the call because the interpretation of privilege flags and file protection code bits depends on the account number of the requester.

In order to use the third party privilege check call, the server must have the correct privilege mask for the requester. The correct way to obtain this information is for the server to issue the Send Message with Privilege Mask subfunction of the Send/Receive SYS call (see Chapter 8). The monitor inserts the requester's current privilege mask as part of the message and sends it to the server, with a message code of -11 to indicate the mask is present. The server then stores the requesting PPN and privilege mask, both from the message, and uses them later in the third party privilege check call.

Server programs could use other methods to obtain privilege information, such as reading the authorized privileges of the account from the accounting data stored on disk. However, this method does not work in all cases. For example, the request may have come from a privileged program. In this case, the privileges of the program, not those of the user, are the relevant ones. Or, the user may have turned off some privileges with the SET JOB/PRIVILEGE command. In this case, the server should honor the lesser privileges.

Check Access Function

This SYS call performs three privilege checking functions:

- Check file access rights. You can use this subfunction to check access rights to a file of known protection code and PPN. You can also use this subfunction to define file-like access rules for objects other than files.
- Convert privilege name to mask. You can use this subfunction to convert a privilege name to its internal representation or to determine whether a user has a specific privilege.
- Convert privilege mask to name. You can use this subfunction to generate the symbolic form of a privilege mask.

Check File Access Rights

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(32%), the check access code.
3	CHR$(0%), the check file access rights code.
4	Reserved; should be 0.
5-6+	PPN.
7-21	Reserved; should be 0.
22+	Protection code of the file.
23-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1-2	Not used.
3-4	The access flags. The bits are set to indicate access rights as follows.

Bit	Meaning
0	Create/rename rights are granted
1	Read access is not allowed
2	Write access is not allowed
3-4	Reserved
5	Execute access is not allowed
6	Accounting rights are granted, or PPN is your own. See Discussion.
7	Accounting rights are granted. See Discussion.
8-15	Reserved
5-30	Not used.

Privileges Required

None.

Possible Errors

No errors are possible.

Discussion

This subfunction checks access rights to a file of known protection code and PPN without opening it.

If you need to check a file's access rights but do not know the protection code, or if you need to read or write to the file, the most straightforward method is to simply open it instead, and then check for an error on open or the read/write access flags returned in the STATUS variable. See the *BASIC-PLUS Language Manual* for a description of the STATUS variable.

You can also use this subfunction to define file-like access rules for objects other than files. For example, Print/Batch Services (PBS) uses it to control access to jobs.

The difference between bits 6 and 7 in the returned flags is that bit 6 is set if the PPN matches the caller's without any privilege requirements, whereas bit 7 is set only if the caller has GACNT or WACNT privileges, even if the PPN is the caller's.

Convert Privilege Name to Mask

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(32%), the check access code.
3	CHR$(1%), the convert privilege name to mask code.
4-6	Reserved; should be 0.
7-12	The privilege flag name. This must be a six-character uppercase ASCII string. For flag names of fewer than six characters, fill the extra space at the end with nulls. Specify ALL to indicate all privileges.
13-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1-2	Not used.
3	Flag byte. 0 if the job currently has the specified privilege, or the job has all privileges if ALL was specified in bytes 7-12. Otherwise, 1.
4-6	Not used.
7-14	A privilege mask with one bit set. If you specified ALL in bytes 7-12, the call returns a privilege mask with all valid bits set (all privilege bits that currently have meaning).
15-30	Not used.

Privileges Required

None.

Convert Privilege Name to Mask
F0=32

Possible Errors

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT The privilege name passed in bytes 7-12 is not a valid privilege name.	5

Discussion

This call performs two functions:

- Converts privilege names to their internal representation. This is useful, for example, when issuing SYS call 28, Set/Clear/Read Current Privileges.
- Determines whether a user has a given privilege. A system program might do this either to verify that a user has sufficient privilege to proceed or to allow the user additional choices.

Convert Privilege Mask to Name

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(32%), the check access code.
3	CHR$(2%), the convert privilege mask to name code.
4-6	Reserved; should be 0.
7-14	Privilege mask.
15-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1	The current job number times 2.
2-6	Not used.
7-14	A privilege mask, with the first set bit cleared (the bit for which the name is returned in bytes 15-20). Unused bits are also cleared.
15-20	The privilege name, as an uppercase ASCII string, padded with nulls to six characters.
16-30	Not used.

Privileges Required

None.

Convert Privilege Mask to Name
F0=32

Possible Errors

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT The privilege mask passed in bytes 7-14 is zero, or no defined privilege bits are set.	5

Discussion

This call scans the privilege mask passed in bytes 7-14. First the call clears undefined bits, then it looks for a set bit. If none are found, it returns the error ?Can't find file or account (ERR=5). Otherwise, it clears the first bit found and looks up its name. The call returns the name of the privilege in bytes 15-20.

A program can use this function to generate the symbolic form of a privilege mask simply by copying the mask into bytes 7-14, and repeatedly issuing this subfunction until the error message is returned. Each time the function returns success, the caller prints out the string in bytes 15-20.

Open Next Disk File

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(33%), the open next disk file code.
3	The channel number times two.
4	Reserved; should be 0.
5-6+	PPN of the file's owner. A value of zero indicates the current account. The specification cannot contain wildcards.
7-10+	File name in Radix-50 format. The specification can contain wildcards.
11-12+	File type in Radix-50 format. The specification can contain wildcards.
13-16	Reserved; should be 0.
17-18	CHR$(N%)+CHR$(SWAP%(N%)), where N% specifies one of the following OPEN modes:

Value	Mode
0%	Normal read/write.
1%	UPDATE mode.
2%	APPEND to file.
5%	Guarded UPDATE (4%+1%).
8%	Special extend.
16%	Do not update access dates to files; do not grant write access (requires DATES privilege).
32%	Do not grant any access to files (directory lookup only).
256%	User data caching.
2048%	Sequential data caching.
4096%	Read normally regardless.
8192%	Open file read only.
16384%	Include files marked-for-delete.
32767%+1%	Mode bits are real; you must specify this value for the other OPEN bits to be examined.

See Chapter 1 for more information about OPEN modes.

19-22	Reserved; should be 0.
23-24+	Device name; must be a disk. A zero in both bytes indicates SY: (the public structure). If you do not specify a name, SY: is used.
25-26+	Device unit number.
27-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1-3	Not used.
4	File size (MSB).
7-10+	File name in Radix-50 format.
11-12	File type in Radix-50 format.
13-14	File size (LSB).
15-16	Date of last access.
17-18	Date of creation.
19-20	Time of creation.
21	File cluster size.
22	File protection code.
23-24	Device name.
25-26	Device unit number and flag.
27-28	File identification index.
29-30	Device description.

Privileges Required

None	Access your own file, or a file in another account if the protection code permits
GREAD	Read a file in any account within the group

WREAD	Read a file in any account
GWRITE	Write a file in any account within the group
WWRITE	Write a file in any account
DATES	Use mode bit 16% to suppress updating of last access date
DEVICE	Access a restricted device

Possible Errors

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT No more files match the passed specification. The channel is closed.	5
?ILLEGAL SYS () USAGE The parameters passed in the call are inconsistent with currently open channel.	18
?DISK PACK IS LOCKED OUT The disk pack is locked, and you do not have the DEVICE privilege.	22
?DEVICE NOT FILE-STRUCTURED You tried to open a device that is not a disk.	30

This SYS call also returns device-dependent errors, such as ?Device hung or write locked (ERR=14) and ?Disk pack is not mounted (ERR=21).

Discussion

This SYS call opens a disk file or a series of disk files matching a wildcard specification. The call requires an I/O channel to use, and grants access to the file if so desired.

When you specify a closed channel, this call finds the first file that matches the specification. When you specify an open channel, the call finds the next file that matches the specification. If there are no more files to find, this call closes the channel.

Note BASIC-PLUS cannot access the channel, even though it is open. To access the channel, use a MACRO subprogram (see the *RSTS/E System Directives Manual*).

Set Device Characteristics and System Defaults

- Set Device Characteristics
- Set Line Printer Characteristics
- Set System Defaults
- Load Monitor Overlay Code and Return Status/Remove Monitor Overlay Code

Set Device Characteristics

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(34%), the set device characteristics or change system defaults code.
3	CHR$(0%), the set device characteristics subfunction code.
4	Reserved; should be 0.
5	CHR$(E%+P%+L%), the flags to indicate changes to device parameters:

E% ENABLED/DISABLED status change flag.

E% = 0% No change.
E% = 1% Use value in next byte to change status.

P% RESTRICTED/UNRESTRICTED device ownership change flag.

P% = 0% No change.
P% = 2% Use value in next byte to change ownership.

L% LOCAL/MODEM keyboard (KB:) control change flag.

L% = 0% No change.
L% = 4% Use value in next byte to change KB: modem control.

You can use combinations of the above values.

6 CHR$(E%+P%+L%), the values you use to change device parameters. If byte 5 is 0%, then this byte must also be 0%.

E% ENABLE/DISABLE flag.

E% = 0% Enable device.
E% = 1% Disable device.

P% RESTRICTED/UNRESTRICTED flag.

P% = 0% No privilege needed for device ownership.
P% = 2% DEVICE privilege needed for device ownership.

L% LOCAL/MODEM keyboard (KB:) control.

L% = 0% No MODEM control
L% = 4% Enable MODEM control.

You can use combinations of the above values.

7-22 Reserved; should be zero.

23-24+ Device name on which to perform the operation. The device cannot be a disk.

25+ Device unit number.

26+ Unit number flag. Should be CHR$(255%) to indicate the device unit number is real.

27-30 Reserved; should be zero.

Data Returned

Bytes	Meaning
1-5	Not used.
6	CHR$(E%+P%+L%), the bit flags indicating device status.

E% = 0 if device enabled and free; 1 if disabled or in use.

P% = 2 if device ownership requires privilege; 0 if not.

L% = 0 if keyboard is LOCAL; 4 if under MODEM control.

7 CHR$(N%), where N% is the job number times 2 of current device owner. If N%=0%, the device is enabled and free. I: N% is an odd integer (other than 3%), the device was

disabled by the monitor and cannot be reenabled. If N% is 3%, the device was disabled by this call and can be enabled by this call.

8-30 Not used.

Privileges Required

HWCFG Change restricted or modem control flags

HWCTL Enable or disable a device

Possible Errors

Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE You attempted to disable a device that was either in use or had been previously disabled. Or, you tried to alter the local/modem characteristic of a device that was in use.	3
?DEVICE NOT AVAILABLE You attempted to enable a device that was not disabled through the use of this call. For example, the monitor disabled the device. Or, you tried to alter the local/modem characteristic of a disabled device.	8
?PROTECTION VIOLATION You attempted to alter the device characteristics of a disk unit.	10
?ILLEGAL SYS( ) USAGE You attempted to perform an invalid subfunction by specifying a value of less than 0 or greater than 3 in byte 3 of the data passed.	18

Discussion

This subfunction allows a caller with the appropriate privilege to set the following device characteristics online:

- Enable or disable a device.
- Designate a keyboard as local or modem.
- Designate a device as restricted or unrestricted.

Set Line Printer Characteristics

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(34%), the set device characteristics or change system defaults code.
3	CHR$(1%), the set line printer (LP) characteristics subfunction code.
4	Reserved; should be 0.
5	CHR$(N%), where N% is: **Value** **Meaning** 0 No change. 1-254 New value for the default printer page width.
6	CHR$(N%), where N% is: **Value** **Meaning** 0 No change. 1-255 New value for the default printer form length.
7-8	CHR$(N%), where N% is 0 for no change or is the combined value of bits to set in the characteristics flag word. See the Discussion for a description of these bit flags.
9-10	CHR$(N%), where N% is 0 for no change or is the combined value of bits to clear in the characteristics flag word. See Discussion.
11	CHR$(N%), where N% is 0 for no change or nonzero to indicate a change to the line printer special character.
12	If byte 11 is CHR$(1%), then this byte is CHR$(N%), where N% is the ASCII value of the new special character. Note that CHR$(0%) disables special character handling. The /SPECIAL_CHARACTER qualifier of the SET TERMINAL command uses this feature. See the *RSTS/E System Manager's Guide* for more information.

13-22	Reserved; should be 0.
23-24+	Device name in two ASCII characters. Must be LP.
25+	Device unit number.
26+	Unit number flag. Should be CHR$(255%) to indicate the device unit number is real.
27-30	Reserved; should be 0.

Data Returned

Bytes	Meaning
1-4	Not used.
5	Width of the line printer unit specified in bytes 23-26 in Data Passed.
6	The default printer form length.
7-8	Current characteristics flag word. See the Discussion for a description of these bit assignments.
9-11	Not used.
12	Value of the line printer special character. The /SPECIAL_CHARACTER qualifier of the SET TERMINAL command uses this feature. See the *RSTS/E System Manager's Guide* for more information.
13-30	Not used.

Privileges Required

HWCFG

Possible Errors

Meaning	ERR Value
?NOT A VALID DEVICE You attempted to set the characteristics of a printer that the monitor does not support, or the device name specified in bytes 23-24 was not LP.	6

?ILLEGAL SYS() USAGE 18
You attempted to perform an invalid subfunction by specifying a value of less than 0 or greater than 3 in byte 3 of the data passed.

%ILLEGAL NUMBER 52
You attempted to set a value of 3 in the LPTCHR flag word. See the Discussion for a detailed description of this error condition.

Discussion

This subfunction allows a caller with HWCFG privilege to set the following line printer characteristics online:

- Change the default page width
- Change the default form length
- Change or read the line printer characteristic flag word.
- Change or read the line printer special character.

Bytes 7-8 set bits in the characteristics flag word. Bytes 9-10 clear bits in the characteristics flag word. Legal values are the following:

Value	Meaning
1%	Allow BS for backspace (LA180, LN01)
2%	Do not process BS as backspace
4%	Allow 8-bit characters (LN01)
8%	Allow nonprinting characters (LN01)
16%	No fill for FF
32%	Allow EOT
64%	No CR required before LF, VT, FF (LP11, LN01)
128%	Ignore CR if next character is LF (LP11, LN01)
256%	No TAB expand (LN01)
512%	Reserved
1024%	Reserved
2048%	Allow lower case (LN01)

Note that a value of 3 in the bottom two bits of the characteristics word is illegal. If you attempt to set both bits, the call returns the error %Illegal number (ERR=52).

Set System Defaults

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(34%), the set device characteristics or change system defaults code.
3	CHR$(2%), the set system defaults subfunction code.
4	Reserved; should be 0.
5-6	CHR$(N%) + CHR$(SWAP%(N%)), where N% is:

Value	Meaning
0%	No change.
1%-300%	New value for powerfail delay.

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

Value	Meaning
0%	No change.
1%	Numeric date format (yy.mm.dd).
255%	Alphabetic date format (dd-mmm-yy).

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

Value	Meaning
0%	No change.
1%	24 hour time format (hh:mm).
255%	12 hour time format (hh:mm AM/PM).

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

Value	Meaning
0%	No change.
1%	DOS magnetic tape label default.
255%	ANSI magnetic tape label default.

10	Reserved, should be zero.
11-12	Default magnetic tape density in bpi.
13-30	Reserved; should be zero.

Data Returned

Bytes	Meaning
1-4	Not used.
5-6	Number of seconds to delay on powerfail restarts.
7	1 if Numeric date format is the default; 255 if Alphabetic.
8	1 if 24 hour time format is the default; 255 if 12 hour.
9	1 if DOS magnetic tape labeling is the default; 255 if ANSI
10	Not used.
11-12	Magnetic tape density in bpi.
13-30	Not used.

Privileges Required

HWCFG	Set the density default
SWCFG	Set the date format, time format, label default, or powerfail delay

Set System Defaults
F0=34

Possible Errors

Meaning	ERR Value
?ILLEGAL SYS() USAGE You attempted to perform an invalid subfunction by specifying a value of less than 0 or greater than 3 in byte 3 of the data passed.	18
%INTEGER ERROR You attempted to specify an value greater that 300 for the powerfail delay value in bytes 5-6.	51

Discussion

This subfunction allows a caller with the appropriate privilege to set the following system defaults online:

- Date format
- Time format
- Magnetic tape label
- Magnetic tape density
- Powerfail delay

The DCL SET SYSTEM command uses this call. See the *RSTS/E System Manager's Guide* for more information on system defaults.

Load Monitor Overlay Code and Return Status/Remove Monitor Overlay Code

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(34%), the set device characteristics or change system defaults code.
3	CHR$(4%), the load/remove monitor overlay code subfunction code.
4	Reserved; should be 0.
5	CHR$(N%), where N% is: 0% Load monitor overlay code and return status 1% Remove monitor overlay code
6	Reserved; should be 0.
7-8	The internal overlay name in Radix-50. The following are defined overlay names: LIN SYS call -25; Manipulate File, Pack, and Account Attributes UUO SYS calls -3, -12, -29; Get Monitor Tables, Parts I, II, III. SYS call -8; Get Open Channel Statistics. SYS call 20; Convert Date and Time. SYS call 26; Return Job Status. SYS call 9; Return Error Messages. DLN The RENFQ and DLNFQ subfunctions of the CALFIP monitor directive. This code handles file delete and rename operations. See The *RSTS/E System Directives Manual* for details. DIR SYS calls 15, 17; Directory Lookup. PFB Miscellaneous functions used in indirect command file processing and DCL. TRM SYS call 16; Set Terminal Characteristics.
9-30	Reserved; should be 0.

Load/Remove Monitor Overlay Code
F0=34

Data Returned for Load/Return Status

Bytes	Meaning
1-4	Not used.
5-6	Amount of XBUF used for overlay, in bytes. 0 if overlay is not loaded.
7-30	Not used.

Data Returned for Remove

No meaningful data is returned.

Privileges Required

SWCFG

Possible Errors for Load/Return Status

Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE The overlay you specified is already loaded	3
?CAN'T FIND FILE OR ACCOUNT The overlay name is not valid.	5
?NOT A VALID DEVICE The overlay you specified is not loadable.	6
?PROTECTION VIOLATION You don't have SWCFG privilege.	10
?NO BUFFER SPACE AVAILABLE There is not enough available extended buffer space (XBUF) to load the overlay.	32

Possible Errors for Remove

Meaning	ERR Value
?CAN'T FIND FILE OR ACCOUNT The overlay name is not valid.	5

?PROTECTION VIOLATION 10
You don't have SWCFG privilege.

?DISK PACK IS NOT MOUNTED 21
The overlay name you specified is not loaded.

Discussion

This subfunction allows a caller with SWCFG privilege to make certain monitor overlay code memory resident. This can enhance system performance if the code is frequently used.

The SYS calls grouped in overlay code UUO return monitor, file, and job information. You should make these calls memory resident in almost all cases.

The system uses file delete/rename code (overlay code DLN) whenever you delete and rename files. Make this code memory resident if your system is large or if your applications require a large number of file delete and rename operations.

The directory lookup code (overlay code DIR) gathers information about disk directories, performs wildcard disk file lookups, and manipulates file identification blocks for certain files. The CATALOG command in BASIC-PLUS and the PIP.SAV program use this code to obtain directory information. The DCL DIRECTORY and COPY commands also use this code. Make this code memory resident if you frequently use any of these programs or commands.

The attribute code (overlay code LIN) performs file attribute read/write operations. Make this code memory resident if:

- You plan to use languages such as COBOL-81, BASIC-PLUS-2, and FORTRAN-77.
- You plan to use the Task Builder.
- You plan to use RMS-11.

The indirect command file processing code (overlay code PFB) performs miscellaneous functions related to ICFP. Make this code memory resident if your system frequently uses indirect command file processing.

The set terminal code (overlay code TRM) sets and reads terminal characteristics. The SET TERMINAL command also use this code. Make this code memory resident if your applications use features that frequently reset terminals (for example, private delimiters).

The internal overlay names are subject to change in future releases. In addition, the actual function performed by the overlays may change, without a corresponding name change.

The DCL LOAD/OVERLAY and UNLOAD/OVERLAY command uses this call. See the *RSTS/E System Manager's Guide* for more information about these commands.

The PEEK Function

The PEEK function lets a user with RDMEM privilege 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.

Note

> When you use the PEEK function, be aware that DIGITAL reserves the right to change the monitor structure and internal addresses at any time, except those addresses listed in Table 7-9. In addition, accessing some device registers can cause unpredictable system results. Do not use PEEK to examine device registers. To protect against this, PEEK requires SYSMOD privilege if the specified address is within the device register address range (160000 octal or higher).

The PEEK function has the form:

```
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. Because addresses of word locations are always even on the PDP-11 computer, and odd addresses indicate byte locations, you must always be careful to specify an even integer address as the argument to PEEK. To examine an odd address, you 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.

You normally use PEEK to examine either addresses returned by Get Monitor Tables calls or addresses of fixed monitor locations.

Possible Errors

Meaning	ERR Value
?PROTECTION VIOLATION A user without RDMEM privilege attempted to execute this call.	10

?ODD ADDRESS TRAP 33
The address specified as an argument to PEEK is odd or an attempt is made to reference a nonexistent or odd address. (For the PDP-11/23 and 11/24, this error occurs only if nonexistent addresses are referenced.)

?MEMORY MANAGEMENT VIOLATION 35
The address specified as an argument to PEEK is illegal (not mapped in the monitor).

Fixed Locations in Monitor

The information shown in Table 7-9 is stored in fixed locations in the monitor part of memory and is obtained by executing a PEEK(X%), where X% is the address shown.

Table 7-9: Monitor Fixed Locations

Address (decimal)	Name	Meaning
36(word)	IDATE	The date when the system was last started.
38(word)	ITIME	The time of day when the system was last started.
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).

Table 7-9: Monitor Fixed Locations (Cont.)

Address (decimal)	Name	Meaning
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).

Finding the Current PPN

Two methods exist for a program to determine the PPN under which it is running. The first method, available to all users, is to execute the Return Job Status SYS call (SYS 26).

The second method, available only to users with RDMEM privilege, is slightly 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 PPN 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 puts the project-programmer word into the variable A%:

```
A% = PEEK(PEEK(PEEK(520%)+8%)+24%)
```

The following statements put the project number in B% and the programmer number in C%:

```
B% = SWAP%(A%) AND 255%
C% = A% AND 255%
```

Chapter 8

System Calls for Local Interjob Communication

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 (with 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 (with the Declare Receiver call).

The system queues messages until the maximum number of messages specified by the receiver is pending for that particular receiver. After that, if a local job tries to send another message to that receiver, the system returns an error to the sender. In general, receivers must process pending messages frequently to avoid tying up system resources for long periods of time. When message processing is complete, a job must issue a remove receiver system call so that unwanted messages are not queued.

DECnet/E network communication uses extensions to the SYS calls presented in this chapter. DECnet/E is an optional software package that extends RSTS/E to include network capabilities. You can access the extended send/receive facilities provided by DECnet/E from BASIC-PLUS, BASIC-PLUS-2, COBOL, FORTRAN, or MACRO, and through RMS. See the DECnet/E and RMS documentation for more information.

If the system manager includes DECnet/E during system installation, 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. It also provides additional capabilities not provided by the local send/receive functions.

Some parameters or combinations of parameters in the send/receive calls have meanings that pertain to special applications, such as an EMT logger, which is a special program that monitors certain types of system activity. These parameters are mentioned briefly in this chapter. See Appendix G for more information on EMT logger calls.

Every message is divided into a parameter area and a data area, whether it is for local or for network communications. 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 data defined by the sender, the distinction between parameter and data is arbitrary for local messages. However, the distinction is important for a network message, in which DECnet/E uses the parameter area for DECnet/E information.

Format of the Send/Receive SYS Calls

The general format of the SYS 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$(S%)	is the user-specified subfunction code. (For example, S%=1% indicates a declare receiver and S%=0% indicates a remove receiver system call).
...	indicates other arguments that must be specified for the system calls. Byte arguments have the form CHR$(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. Word arguments have the form CVT%$(SWAP%(X%)).
O$	is optional user-defined data.

The system call descriptions use the following terms:

Term	Meaning
Reserved; should be zero.	This field is reserved for future use. You must specify zero for each byte in the field. Trailing zeros need not be passed.
Not meaningful; should be ignored.	The bytes in this field do not contain useful information. However, these bytes may have meaning in future releases. This term appears in the data returned by the various SYS calls.

Note

Unlike the SYS calls to FIP (see Chapter 7), the arguments passed to and returned from this Send/Receive call are longer than 30 bytes. You should dimension the arrays used in CHANGE statements to handle 40-byte strings (see the sections, "Building a Parameter String" and "Unpacking the Returned Data," in Chapter 7).

Privileges Required for Send/Receive

SEND	Send to a restricted receiver
SYSIO	Declare a receiver with a nonzero Local Object Type, global name, or nonzero inbound links limit
JOBCTL	Remove the RIB of another account

F0=22
(.MESAG)

Declare Receiver

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; should be 0.
5-10	The receiver name. The receiver name must be a one- to six-character ASCII string. It must be left-justified and padded to six characters with spaces. The receiver name must contain only printing ASCII characters (characters with ASCII decimal values in the range 33 to 126) and cannot contain leading or embedded spaces. However, you can specify a blank receiver name (six spaces). If you do not have SYSIO privilege and you specify a nonblank name, it must contain six characters and the last two characters must be your job number, as two ASCII digits.
11-20	Reserved; should be 0.
21	CHR$(O%), the object type code. Legal values are 0 through 255. Object type codes for network send/receive are defined in *DECnet/E Network Programming in BASIC-PLUS and BASIC-PLUS-2*. See the Discussion for more information on object type codes for local receivers.
22	CHR$(L%+P%+N%+O%+S%), Access Control Field. This byte controls the types of senders that are allowed to queue messages for this job and also controls system handling of queued messages. It is the sum of the following five bit values: L% Local/No Local Senders

If L%=0%, messages from local senders are not queued. Local senders who use network functions are considered network senders in this context.

If L%=1%, messages from local senders are queued.

P% Local Privileged/Local Nonprivileged

This bit is ignored if L%=0% (no local senders allowed).

If P%=0%, local senders without the SEND privilege can queue messages.

If P%=2%, local senders must have the SEND privilege.

N% Network Logical Links/No Logical Links

This bit controls the queuing of requests for DECnet/E logical links. The *DECnet/E Network Programming in BASIC-PLUS and BASIC-PLUS-2* manual describes network links. For local interjob communication, this bit should be zero.

If the receiver does not have SYSIO privilege, this bit must be zero.

O% Network Single Links/No Single Links

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

S% Sleep/No Sleep

If S%=0%, pending messages block execution of a conditional SLEEP. (In a conditional SLEEP, the monitor checks for certain conditions before executing the SLEEP statement. One of these conditions is pending messages in the job's message queue. See the section, "SLEEP and Conditional SLEEP Statements," in Chapter 10, for more information.)

If S%=16%, pending messages do not block execution of a conditional SLEEP.

23-24 Reserved; should be 0.

25 CHR$(M%), the message maximum.

The message maximum can be any value between 1 and 255. However, if the job does not have EXQTA privilege, this value cannot exceed the value in the account's second quota block. See Discussion.

26 CHR$(L%), the inbound link maximum.

The incoming link maximum declares the maximum number of incoming DECnet logical links a job will support at any one time. For local interjob communication, the incoming link maximum should be 0. If you have SYSIO privilege, the incoming link maximum can have any value between 0 and 255. If the receiver does not have SYSIO privilege, this value must be zero.

27-28 Reserved; should be 0 for all receivers except an EMT logger. See Appendix G for more information.

29 CHR$(O%), the outbound link maximum.

DECnet uses this parameter as the number of outgoing links. If you have EXQTA privilege, the outgoing link maximum can have any value between 0 and 255. If you do not have EXQTA privilege, this value must be 0 or 1. (Note that in either case, a value of 0 is interpreted as a default to the maximum value.)

30 Reserved; should be 0 for all receivers except an EMT logger. See Appendix G for more information.

31-34 CHR$(0%), reserved; should be 0.

35 CHR$(R%), the receiver ID block (RIB) number. It must be a value between 0 and 255; values 128 through 255 are reserved for use by DIGITAL.

A job that does not have EXQTA privilege can use only the number of RIBs specified in the account's second quota block.

36-40 CHR$(0%), reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

SYSIO — Declare a receiver with a nonzero Local Object Type, global name, or nonzero inbound links limit

Possible Errors

Meaning	ERR Value
?ILLEGAL FILE NAME One of the following occurred: o The receiver name passed in bytes 5-10 contains nonprinting ASCII characters or leading or embedded spaces. o A job that did not have SYSIO privilege passed a nonblank receiver name that does not contain its job number in bytes 9 and 10. o The specified local object type code is invalid.	2
?ACCOUNT OR DEVICE IN USE The calling job already exists in the receiver's list of declared receivers for the specified receiver ID block (RIB) number. This error may indicate that the program contains a logic error or that a previous program running under the same job number failed to remove itself from the receiver list before terminating. In the last case, the calling job should remove itself (with a Remove call, see the section "Remove Receiver").	3
?PROTECTION VIOLATION One of the following occurred: o The specified RIB number is out of range. o The caller does not have sufficient privilege at the time certain functions were attempted in the receiver declaration. (For example, a caller needs SYSIO privilege to declare "single-instance" local object type.)	10
?NAME OR ACCOUNT NOW EXISTS The receiver name passed in bytes 5-10 is already being used as a receiver ID (this error cannot occur if the name specified is blank), or a specified "single-instance" local object type is already in use.	16

?ILLEGAL SYS() USAGE 18
The receiver name, object type, and access parameters passed are inconsistent.

?ILLEGAL BYTE COUNT FOR I/O 31
The value you specified in byte 30 is out of range (refers to an EMT logger; see Appendix G for more information).

?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. Since the system's use of small buffers is dynamic, a retry may succeed.

?MISSING SPECIAL FEATURE 66
The call you attempted requires an optional feature (such as EMT logging) that is not available on your system.

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 at least one receiver ID block.

Using Multiple Receiver ID Blocks

A single job can declare itself as more than one receiver by executing multiple Declare Receiver calls. Each job has available for its use 256 receiver ID blocks (RIBs), numbered 0 through 255. RIB numbers 0 through 127 are for customer use; RIB numbers 128 through 255 are reserved for use by DIGITAL. You specify the RIB number in byte 35; if you do not want to use multiple RIBs, set this byte to zero.

The number of RIBs assigned to jobs that do not have EXQTA privilege are counted and checked against a quota. The quota is stored in the account's second quota block.

For each successful declare, the system sets up a separate 32-byte RIB. Each RIB has a separate message queue and can have different characteristics (for example, message maximum and type of senders allowed). You specify these characteristics in the call. The system allocates space for each RIB from the small buffer pool.

Multiple RIBs can be useful in a program that performs several functions. For example, in certain applications, you might want to use one RIB for high priority messages or messages from specially privileged senders, and another for lower priority messages or messages from any sender. If the program consists of several subroutines or modules, each one can process messages independently using different RIBs.

Receiver Names

You must associate each RIB that you declare with a receiver name. Place the receiver name in bytes 5 through 10 of the data passed. While your program needs to keep track of both the RIB number and the receiver name, other jobs can send messages to you by using either the receiver name or the job number.

A receiver name must be either unique or blank (six spaces). Receivers that receive messages by job number rather than by name use blank receiver names. If you do not have SYSIO privilege, you can use blank names. Any nonblank names you use must have six characters, and the last two characters must be the job number. Privileged programs may use other names. However, when multiple copies of a program are active at the same time, DIGITAL recommends that privileged programs also use the job number as the last two characters of the receiver name to avoid name conflicts.

Note

To avoid future name conflicts, do not use the $ character in your own receiver names.

Table 8-1 lists the names currently used by DIGITAL software.

Table 8-1: RSTS/E Reserved Names

Reserved Name	Local Object Type	Use
ERRLOG	1	Error logger
BAnSPL	--	OPSER batch processor
LPnSPL	--	OPSER print spooler
OPSER	--	OPSER message manager
QUEMAN	6	OPSER queue manager
QM$CMD	3	PBS spooling package
QM$SRV	4	PBS spooling package
QM$URP	5	PBS spooling package
PR$nnx	65	PBS spooling package
BA$nnx	66	PBS spooling package
SHUTUP	--	System shutdown program
EVTLOG	--	DECnet/E
EVTLSN	--	DECnet/E
FALnnn	--	DECnet/E
NCPnnn	--	DECnet/E
NMLnnn	--	DECnet/E
NWPKnn	--	DECnet/E
NWTTnn	--	DECnet/E

To see which receiver names are reserved on your system, use the SHOW RECEIVERS command.

For example:

```
$SHOW RECEIVERS

Message Receivers:
Rcvrid    Job    Rib   Obj    Msgs/Max    Links/InMax/OutMax   Access
ERRLOG     1       0     1       0/40           0/0/0            Prv
OPSER      2       0     0       0/30           0/0/255          Lcl
   .
   .
   .
```

Local Object Types

A receiver ID block can be associated with an object type (specified in byte 21). Object type codes for network send/receive are defined in the DECnet/E documentation.

If you have SYSIO privilege, you can specify a local object type for receivers that do not perform network operations. The code you specify indicates that a receiver performs a specific function. For example, the error logger, which records error information for the monitor, is assigned local object type 1. If your system uses an EMT logger, it is assigned local object type 2. You can also specify local object type 64, which allows error messages to be sent to your program if you open a disk non-file-structured with MODE 512% (see the section, "Access to Bad Block Information" in Chapter 1).

Local object types 7 through 63 are "single-instance" local objects used by DIGITAL-supplied programs. The system makes sure that no more than one receiver of each single-instance type is declared, allowing the system to rapidly locate a given receiver for certain functions.

Access Control Field

Byte 22 controls the types of network access permitted and the types of local senders permitted. The possible values are 0-31. However, for local interjob communication, only the following values are allowed:

1 Any local sender

3 Only senders that have SEND privilege

17 Any local sender; sleep with pending messages

19 Only local senders that have SEND privilege; sleep with pending messages

The "sleep bit" (value 16%) in the access control field is mainly for use with multiple RIBs. When this bit is set, pending messages do not block execution of a conditional SLEEP (see the section "SLEEP and Conditional SLEEP Statements" in Chapter 10). By setting this bit, you can process messages on a specific RIB only when your program is executing certain code and, at the same time, prevent pending messages on that RIB from affecting conditional SLEEP statements in the rest of your program.

Buffer Space for Messages

Each pending message in the system occupies system buffer space. Except for EMT logger messages, 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 allocates buffer space for messages from the extended buffer pool (XBUF).

Queued Message Limit

The system maintains a count of messages queued for each receiver. The message maximum (byte 25) limits the number of messages queued for this receiver. This limit applies both to messages from local senders and network data messages. Local messages and network data 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 reached this maximum.

The declared value for the message limit is constrained by the account's message quota, which is stored in the account's record quota block. The sum of the message limits for all the receivers a job has declared cannot exceed the message quota. For example, if you have a message quota of 12 (the default), you can declare one receiver with a message limit of up to 12, or one with a limit of 3 and another with a limit of up to 9. If the job has EXQTA privilege in effect, the system ignores the message quota.

F0=22
(.MESAG)

Send Local Data Message

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%). If J%=0, the call uses the logical name in bytes 5 through 10 to determine the receiver. If J%=128+LOT, the call uses the local object type (LOT) specified in byte 21. Only single-instance object types are valid. See the Discussion for valid LOTs. Otherwise, J% can be set to the job number times 2 of the local receiving job. For example, specifying J%=8% directs a message to job 4.
5-10	N$, the receiver name. The receiver name is a one- to six-character ASCII string. It is left-justified and padded to six characters with spaces. If byte 4 is nonzero and you specify a receiver name, the send will succeed only if the name in bytes 5-10 is associated with the job number in byte 4.
11	CHR$(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 1-12), 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; should be 0.

13-14 L%, the length (in bytes) of the message to send from the channel buffer in the form CVT%$(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 (that is, from the offset to the end of the buffer).

15-16 O%, the offset value in the form CVT%$(SWAP%(O%)).

The value specifies the offset from the beginning of the buffer where the message data begins. The offset must be in the range zero to <size of buffer - 1>.

17-20 CHR$(0%), reserved; should be 0.

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. The call ignores these bytes if byte 11 is nonzero.

Data Returned

Bytes	Meaning
4	The job number times 2 of the receiving job.

Privileges Required

SEND Send to a restricted receiver

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 allowed by the receiver is too small. This error can also occur if the message receiver is hibernating. Because the hibernating receiver cannot process messages, the system sets a flag for messages sent to it and thus minimizes the number of small buffers that would be tied up.	4
?CAN'T FIND FILE OR ACCOUNT For local messages, the receiving job, referenced by job number or logical name, was not found in the list of declared receivers. The receiving job must be declared (with the Declare Receiver SYS 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 An access violation has occurred. Either the sender is nonprivileged and the receiver requires senders to have SEND privilege, 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 31

The offset and/or length fields passed in bytes 13-16 are illegal. The following relationships must be true for a send call:

- The offset must be less than the buffer size.
- 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.

?NO BUFFER SPACE AVAILABLE 32

System buffers are not currently available to store this message. A later retry may proceed without error.

Discussion

A local job can send a message to a declared receiver by specifying:

- A job number
- A logical name
- A logical name, while checking the job number
- A single-instance local object type (if appropriate).

If byte 4 of the data passed is nonzero and even, and the name field (bytes 5-10) is null, the call interprets it as the job number (times 2) of the intended receiver.

If bytes 5-10 are not null, the call attempts to send the message to the receiver whose logical name matches bytes 5-10 of the data passed. If byte 4 is zero, the message is sent to the named receiver. If byte 4 is nonzero and even, the message is sent to the named receiver only if that receiver is owned by the job whose job number times two is specified in byte 4. Because it does not require a receiver table search, sending messages by job number or local object type is slightly more efficient than sending by logical name.

If byte 4 is 128 + LOT (local object type), the call sends the message to the receiver designated by the local object type value you specify in LOT. Legal values are:

LOT	Receiver
1	Error logger
2	EMT logger
3	PBS spooling package
4	PBS spooling package
5	PBS spooling package user request packet
6	OPSER-based spooling package

A send by job number works only when the receiving job is receiving messages on RIB 0. (That is, the receiving job must have executed a declare receiver call with a value of zero in byte 35.) All messages sent by job number are queued on RIB 0. If the job is a receiver on one or more nonzero RIBs but not RIB 0, the send by job number fails. In contrast, a send by logical name (with or without job number check) works for any RIB number.

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 fail. Similarly, if local senders must have SEND privilege, an attempt by an insufficiently privileged job to send a message to this receiver also fails. All such access violations terminate with the error ?Protection violation (ERR=10), and the message is not sent.

F0=22
(.MESAG)

Send Local Data Message With Privilege Mask

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(22%), the send/receive function code.
3	CHR$(-11%), the send with privilege mask subfunction code.
4	CHR$(J%). If J%=0, the call uses the logical name in bytes 5 through 10 to determine the receiver. If J%=128+LOT, the call uses the local object type (LOT) specified in byte 21. Only single-instance object types are valid. Otherwise J% can be set to the job number times 2 of the local receiving job. For example, specifying J%=8% directs a message to job 4.
5-10	N$, the receiver name. The receiver name is a one- to six-character ASCII string. It is left-justified and padded to six characters with spaces. If byte 4 is nonzero, the send will succeed only if the receiver name is associated with the specified job number. See Discussion.
11	CHR$(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$(0%), reserved; should be 0.
13-14	L%, the length (in bytes) of the message to send from the channel buffer in the form CVT%$(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 (that is, from the offset to the end of the buffer).
15-16	O%, the offset value in the form CVT%$(SWAP%(O%)). The value specifies the offset from the beginning of the buffer where the message data begins. The offset must be in the range zero to <size of buffer - 1>.
17-28	CHR$(0%), reserved; should be 0.
29-40	P$, the optional user parameter string. You can pass a maximum of 12 bytes of user-defined data as parameters to the receiver of this message.
41+	D$, the optional data string. You can pass a maximum of 512 bytes of user-defined data to the receiver's buffer. The call ignores these bytes if byte 11 is nonzero.

Data Returned

Bytes	Meaning
4	The job number times 2 of the receiving job.

Privileges Required

SEND	Send to a restricted receiver

Possible Errors

This call returns the same errors as the Send Local Data Message call. See the previous section.

Discussion

This call sends a data message plus a privilege mask supplied by the monitor. This subfunction provides a method for a program to tell another program about a job's current privileges and guarantees that the data cannot be falsified.

A local job can send its privileges to a declared receiver in the same way as the Send Local Data Message call. See the Discussion in the previous section.

F0=22
(.MESAG)

Receive

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 Discussion.

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.

In either case, 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. If N% (described as follows) 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.

N% Network Selection

If N%=0%, network selection is disabled. If L% 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% in byte 4.

Any nonzero value in this byte selects a particular sending job. Zero is a special case described for byte 6. See the Discussion for meaningful combinations of bytes 5 and 6.

For local selection (L%=4% as described previously), 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. See the *DECnet/E Network Programming in BASIC-PLUS and BASIC-PLUS-2 Manual* for details.

6 CHR$(Q%), the sender selection qualifier.

This byte is ignored if both L%=0% and N%=0% in byte 4.

This byte is also ignored if byte 5 is nonzero. See the Discussion for meaningful combinations of bytes 5 and 6.

For local selection, if byte 5 is zero and byte 6 is nonzero, 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 or an EMT logger, which receives messages from the monitor's error logging or EMT logging routines. The job number in these messages is zero. See Appendix G for more information on EMT logging.

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 Reserved; should be 0.

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. See Discussion.

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; should be 0.

13-14 L%, the maximum message length (in bytes) desired on this receive in the form CVT%$(SWAP%(L%)).

If byte 11 is zero (that is, 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, the error ?No room for user on device (ERR=4) 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 CVT%$(SWAP%(O%)). The offset field determines where in the buffer the data portion of the message is returned. The offset value is added to the location of the beginning of the buffer. The offset value must be in the range 0 to (size of buffer - 1).
17-20	CHR$(0%) reserved; should be 0.
21-26	Reserved; should be 0.
27-28	T%, the sleep time in seconds in the form CVT%$(SWAP%(T%)). If byte 4 requests a sleep and no messages are pending, the sleep is terminated after T seconds. If T%=0%, the length of the sleep is indefinite; the job is not awakened until one of six events awakens the job. When the sleep terminates, the error ?Can't find file or account (ERR=5) occurs. See Discussion.
29-34	CHR$(0%) reserved; should be 0.
35	CHR$(R%), the RIB number for this receive. The RIB number must be a value from 0 to 255; values 128 through 255 are reserved for use by DIGITAL.
36-40	CHR$(0%) reserved; should be 0.

Data Returned for 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 of the sender.
7	Keyboard number of the sender or 255% if the sender is detached.
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 not delivered to the caller on this receive. If truncation was not requested (T%=0% in byte 4 of the data passed) and not all of the message was delivered, 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 removed from the queue 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 with zeros to a length of 20 bytes.

Data Returned for Local Data Message with Privilege Mask

Bytes	Meaning
1-2	Not meaningful; should be ignored.
3	CHR$(-11%), the local data message with privilege mask subfunction code.
4	CHR$(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 of the sender.

7 Keyboard number of the sender or 255% if the sender is detached.

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 not delivered to the caller on this receive. If truncation was not requested (T%=0% in byte 4 of the data passed) and not all of the message was delivered, 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 removed from the queue 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.

21-28 The sender's privilege mask.

29-40 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 with zeros to a length of 12 bytes.

Privileges Required

None.

Possible Errors

Meaning	ERR Value
?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).	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 on the specified RIB number. Before any receive can succeed on that RIB number, 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: o The offset must be less than the buffer size. o 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

Discussion

On any receive call, the system checks the eligibility of the job to receive messages and returns the error ?Illegal SYS() usage (ERR=18) if the specified job and RIB number are not in the list of declared receivers. If the job is eligible to receive messages, 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. Use the selective receive bits (L% and N% in byte 4) 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 the error ?Can't find file or account (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). You can specify the sleep time in bytes 27-28 of the data passed.

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

- A user types a delimiter (RETURN, LINE FEED, FORM FEED, or ESCAPE) at:
  - Any terminal opened by the job.
  - Any terminal allocated to the job if the job also has a keyboard open on a nonzero channel.
- A dial-up line that is allocated or opened by the job gets hung up.
- The system manager disables logins (that is, sets the number of logins to 1).
- A state change occurs on a pseudo keyboard opened by the job. This condition can occur when the opened pseudo keyboard has output for the controlling job or has entered an input wait state. See the section "Pseudo Keyboards" in Chapter 4.
- The job has declared itself a receiver and a message is queued for it through the Send/Receive SYS calls. See Chapter 8.
- The job has a DMC/DMR (XM:) device open and the device driver receives a message (see Chapter 6).

In all cases, the job is awakened with an error ?Can't find file or account (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,

you can check for pending messages 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.

You can determine 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) 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 kept or discarded. If you set T%=0%, the remaining bytes are kept. 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 removed from the queue.

A convenient way to assign a buffer for message operations is to open the null device (NL:) at the desired buffer size with the RECORDSIZE option in 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 you specify a buffer, the system ensures that the channel is open. If the channel is not open, the call results in an immediate error ?I/O channel not open (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.

Table 8-2 summarizes the possible combinations.

Table 8-2: Sender Selection Summary

Data Passed				Result
Receive Modifier				
Byte 4		Byte 5	Byte 6	
N%	L%			
0%	0%	-	-	Bytes 5 and 6 ignored; returns first queued message.
-	4%	0%	0%	Selects first local message.
		0%	nonzero	Selects job 0; used by error logging and EMT logging programs to select messages from monitor routines.
		nonzero	-	Selects local message by job number times 2 in byte 5.
8%	0%	0%	0%	Selects first network message.
		nonzero	-	Selects network message by link (user link address) in byte 5.

F0=22)
(.MESAG)

Remove Receiver

Data Passed

Bytes	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$(0%) to remove the calling job. If J%=0%, the calling job need not be privileged. If J% is not 0 and not the caller, the caller must have JOBCTL privilege. Add 128% to this byte for "conditional" remove. See Discussion.
5-34	CHR$(0%) reserved; should be 0.
35	CHR$(R%), the receiver ID block (RIB) number to remove. The RIB number must be a value from 0 to 255; values 128 through 255 are reserved for use by DIGITAL.
36	CHR$(0%) to remove the RIB specified in byte 35. A nonzero value in this byte removes all RIBs for the job specified in byte 4.
37-40	CHR$(0%) reserved; should be 0.

Data Returned

No meaningful data is returned.

Privileges Required

JOBCTL	Remove the RIB of another account

Possible Errors

Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE This occurs for a conditional remove if there still are messages pending. See Discussion.	3
?PROTECTION VIOLATION The caller does not have JOBCTL privilege and has attempted to remove another job (that is, byte 4 is nonzero and does not match the caller's job number).	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 receivers for another job.	18

Discussion

This call removes a receiver from the system's list of declared receivers. You can remove all RIBs for a job or a specific RIB for a job. (Be careful when removing all RIBs for a job.) When this call is executed, all pending messages for the receiver are discarded. You should execute this call 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.

Note that both the LOGOUT SYS function and the KILL SYS function execute this call with a nonzero value in byte 36.

The "conditional" remove operation is useful for programs that want to process messages until none remain, then remove the receiver. If you do a receive, find that there are no more messages, and then do a remove, a message could arrive in the time between the receive and the remove; the remove operation would discard that message. To avoid this problem, add 128% to byte 4 for "conditional" remove; in this case, the system returns the error ?Account or device in use (ERR=3) rather than discard messages if some messages are still waiting to be received.

Local Send/Receive Examples

This section gives several examples of the send/receive SYS calls. The examples include a receiver declaration, two send local data calls, five receives to show some of the possible options, and a remove receiver call. The series of examples is a program that 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.

Declare Receiver Example

The following receiver declaration establishes the caller as a message receiver with the logical name "DEMO". Only local senders that have SEND privilege are allowed to send messages to this receiver. Up to five messages are queued for this receiver before senders receive an error (also ERR=4). Finally, no requests for incoming DECnet logical links are 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 SENDERS WITH SEND
                                !PRIVILEGE ALLOWED
1150  MMAX%      = 5%           !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$(ACCESS%)
               + CVT%$(0%)
               + CHR$(MMAX%)
               + CHR$(LMAX%))
```

Send Local Data Examples

The following local send calls 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 you ran this series of examples as a single 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 is delivered in the target string returned by the SYS call and the remainder is 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 sends 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 are 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
```

Receive Examples

This section presents five receive examples. If you ran this series of examples as a program, the receives would retrieve the two messages sent in the send examples of the previous section.

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 because no messages are expected that will be larger than the buffer available. In this example, the ON ERROR GOTO, which is normally 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 rest 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 USABLE
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 determines the sender's job number and length of the next pending message. The call requests an indefinite length sleep 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 this 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 shown. 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 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 USABLE
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). The rest of the message (BYTREM% characters) is still queued. The last receive retrieves the rest of the message and places it in the buffer immediately after the portion delivered on the previous receive. Sender selection makes sure 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 USABLE
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
```

In 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 the previous section also included some "parameters" that were actually delivered to the receiver on each of the last three receives. You can verify this by printing the last 20 characters of the target string returned by the last receive call:

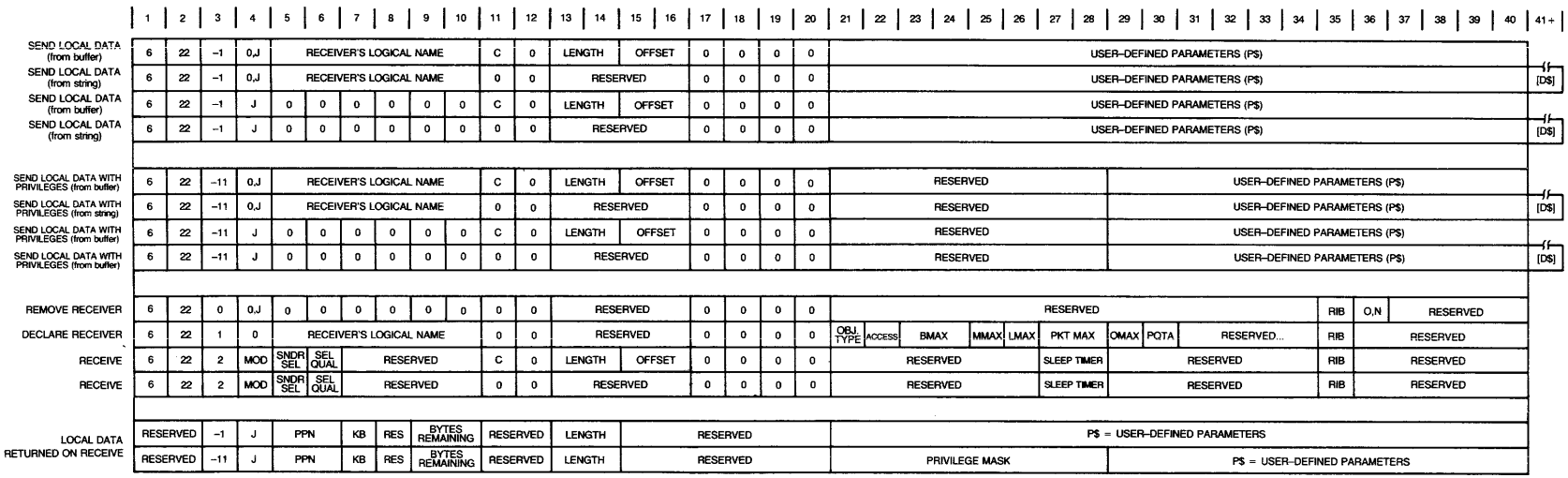
```
4100  !
      ! PRINT PARAMETER AREA OF SECOND MESSAGE FOR VERIFICATION
      !
4110  PRINT "PARAMETER AREA   = ";RIGHT(X$,20%)
```

The example ends 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
```

Summary of Data Values

Figure 8-1 summarizes the data passed and returned in the send/receive calls.



MK-00038-01

Figure 8-1: Summary of Send/Receive Data

Chapter 9

System Call for Print/Batch Services

This chapter describes the System Call for Print/Batch Services (PBS), a subfunction of the Send/Receive System Function Call (SYS 22). This call allows an application program to communicate directly with PBS to issue print or batch requests. This system call does not communicate with the OPSER-based spooling program. Use the Spooling SYS call (SYS -28) to issue requests to OPSER.

In its request, the application can include all of the parameters available with the PRINT and SUBMIT commands. (See the *RSTS/E System User's Guide* for more information on DCL commands.) PBS request processing differs from DCL, however, in that when PBS receives a request, it does not check file names and form names that you specify in the request to see if they exist.

Sending a User Request Packet

An application issues a print or batch request by creating a request packet and sending it to PBS with the Send/Receive subfunction, Send User Request Packet. Unlike the Spooling SYS call (SYS -28), most of the parameters in the request are in the data buffer portion of the message. For the parameters and data that you can include in a user request packet, see the Data Passed and Data Field Layout sections.

Confirming a User Request

Often an application wants to know whether PBS accepted or rejected a request and what error caused the request to be rejected. User request packets allow an application to include the name of a receiver to which PBS returns a completion status message. After PBS processes the request, it returns status information to the designated receiver. On the other hand, an application that does not need confirmation of its request can omit a receiver name from the request.

Send User Request Packet

Data Passed

Bytes	Meaning
1	CHR$(6%), the SYS call to FIP.
2	CHR$(22%), the send/receive function code.
3	CHR$(-11%), the send local message with privilege mask subfunction code.
4	CHR$(128% + 5%). This value indicates a send request for local object type 5. This object type is reserved for the queue manager receiver that processes user request packets.
5-10	Reserved; should be 0.
11	CHR$(C%), the channel number for the I/O buffer that contains the data portion of the message. If this byte is 0%, a string beginning at byte 41 contains the data portion of this message (if any). If this byte contains a channel number (any value from 1 to 12), a buffer defined by the length and offset values contains the data portion of this message. The message data (up to 512 bytes) should be left-justified in the buffer for channel C%, beginning at the offset value defined in bytes 15-16. Channel 0 can be used for the I/O buffer if 128 is added to the channel number, that is, CHR$(128%+0%). In general, CHR$(128%+C%) allows channels 0 through 12 to be used for I/O buffers.
12	CHR$(0%), reserved; should be 0.
13-14	L%, the length (in bytes) of the message to send from the channel buffer in the form CVT%$(SWAP%(L%)). If byte 11 is 0%, the system ignores these bytes. For local data messages, this length field can have any value between 0 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 0, the system sends the whole buffer (that is, from the offset to the end of the buffer).

15-16 O%, the offset value in the form CVT%$(SWAP%(O%)).

The value specifies the offset from the beginning of the buffer where the message data begins. The offset must be in the range 0 to <size of buffer - 1>.

17-28 Reserved; should be 0.

29 CHR$(N%), where N% must be one of the following values:

Value	Meaning
1%	Print request
2%	Batch request

30 Reserved; should be 0.

31-36 N$, the receiver ID for confirmation

The receiver name is a one- to six-character ASCII string containing the name of the receiver to receive a confirmation message. It is left-justified and padded to six characters with spaces. If the first byte is a null, CHR$(0%), then no confirmation message is returned.

The confirmation message either indicates that the request was accepted (success), or contains an error code indicating why the request was rejected and, if applicable, the code of the field that caused the error. See Discussion.

37-38 C$, a confirmation context value. See Discussion.

39-40 Reserved; should be 0.

41 + The data field. See the Data Field Layout section.

Confirmation Data Received by Receiver

Bytes	Meaning
1-2	Not used.
3	CHR$(-11%), the local data message with privileges subfunction code.
4	The job number times 2 of the PBS job that processed the request.
5-6+	PPN of the PBS job.
7	CHR$(255%)

8	Not used.
9	CHR$(0%)
10-28	Not used.
29-30	The confirmation value passed in bytes 37-38 when the request was sent.
31-32	The error status code, indicating whether the request was accepted. If this code is 0, the request was accepted. If nonzero, the request was not accepted, and the value in this field indicates the error code (see the Possible Errors section).
33-34	The field code. For rejected requests, these bytes identify the field in which the error occurred. For successful requests, these bytes contains the entry number assigned to the request.
35-40	Not used.

Privileges Required

EXQTA	Override the queue's maximum limit on priority, pages, time, and CPU time.
GACNT	Specify a different owner within the group.
WACNT	Specify any owner.

Possible Error Codes in the SYS Call

This call returns the same errors as the Send Local Data subfunction of the Send/Receive SYS call. See Chapter 8 for more information.

Possible Error Codes in the Confirmation Message (bytes 31-32)

Meaning	Value
?ILLEGAL FILE NAME For the /NAME, /QUEUE, /FORMS, or Logfile Queue Name field, the name specified contains one or more invalid characters. For the ASCII file-spec, Binary file-spec, or /LOG_FILE field, the field contains an invalid RSTS/E file specification or does not refer to a disk file.	2

?NO ROOM FOR USER ON DEVICE 4

For the /QUEUE field, the specified queue is currently closed or marked for deletion.

For the Logfile Queue Name field, the specified print queue is currently closed or marked for deletion.

?CAN'T FIND FILE OR ACCOUNT 5

For the /QUEUE field, the specified queue does not exist, or no default queue is currently defined.

For the Logfile Queue Name field, the specified print queue does not exist, or no default print queue is currently defined.

For the /OWNER field, the account specified is not a valid RSTS/E account or does not exist. Note that no-user accounts (without a password attribute block) are not considered valid accounts.

For the /FORMS field, no /FORMS field was specified, but the request's print queue does not have a default form name defined.

?PROTECTION VIOLATION 10

For the /QUEUE and /LOG_QUEUE fields, the owner does not have sufficient privilege to use the specified queue.

?NO BUFFER SPACE AVAILABLE 32

There is insufficient buffer space in the PBS package to process the request. This is a temporary condition. Issue the request again after a short delay.

?VIRTUAL BUFFER TOO LARGE 42

The internal entry packet generated by the request exceeded 512 bytes. Reduce the number of file specification fields or reduce the length of the /PARAMETERS field.

?ILLEGAL NUMBER 52

For the /JOB_COUNT field, the job count value specified was not in the range 0-127.

For the /AFTER field, the date or time value specified was not a valid RSTS/E date or time.

For the /PRIORITY field, the priority value exceeded the maximum priority allowed for the request's queue.

For the /PAGE_LIMIT, /CPU_LIMIT, or /TIME_LIMIT field, the limit value exceeded the maximum limit allowed for the request's queue.

?SUBSCRIPT OUT OF RANGE 55
The specified field code was invalid. You can use only the field codes listed in Table 9-1.

?NOT ENOUGH DATA IN RECORD 59
No ASCII (code 128) or binary (code 129) file specification was found. A User Request Packet requires at least one file specification field.

?FIELD OVERFLOWS BUFFER 63
The field was incomplete. This error occurs when the end of the data buffer is encountered before the expected end of a field is reached.

?ARGUMENTS DON'T MATCH 88
If the field code returned is 0, then the request type specified in byte 19 was invalid.

If the field code returned is nonzero, then that field conflicted with the request type specified; that is, a batch-only field was included with a print request, or a print-only field was included with a batch request.

Discussion

Bytes 31-36 contain the name of the receiver to receive a confirmation message. If the first byte is 0, PBS does not return a confirmation message. If a receiver name is specified, PBS makes sure that the receiver name belongs to the job issuing the request. If it does not, PBS does not return confirmation.

Bytes 37-38 contain a confirmation word for the confirmation message. PBS returns this value to the requesting job, allowing it to match a confirmation message with a request message sent. This is useful when an application sends several request packets and needs a separate confirmation for each. In such cases, you should assign a unique context value to each request, so that your program can properly match the confirmation message with its request. you do not specify a confirmation message receiver (byte 31 = 0), PBS ignores this word.

PBS returns the confirmation context value in bytes 29-30, if the caller requested one by specifying a receiver name (other than a null receiver name) in the request. However, there are cases when PBS does not return a confirmation packet, even though the caller specified a receiver name. This occurs when the caller passes the parameter data (bytes 1-40) incorrectly, causing PBS to ignore the request

completely. Also, if the receiver name specified does not exist or does not belong to the job issuing the request, PBS processes the request but does not return a confirmation.

If the user request is successful, PBS places the requested entry on a queue. This remains true even in cases where confirmation fails.

Programs should perform Receive With Sleep functions while waiting for confirmation of a request. See the Receive subfunction of the Send/Receive SYS call in Chapter 8 for more information.

The time required for PBS to complete a request depends on the request itself and the overall system load. It is good programming practice for programs waiting for confirmation to "time out" after a reasonable amount of time (30 seconds is usually sufficient), in case the request was passed improperly.

Data Field Layout

This section describes each of the data fields that you can include in a user request packet. You pass these fields in the data buffer portion of the message, beginning at byte 41. You can also pass the fields in an alternate buffer assigned to a channel number. Each field in the data buffer must begin on an odd byte and consists of a 1-byte field code followed by one or more data bytes for the field. Fields that use word or double-word values require that all word values start on an odd byte.

Some fields have a fixed length, while others include text strings and are variable in length. Variable-length strings always consist of a length byte followed immediately by the characters of the string, called "counted ASCII" strings. The length byte of a counted ASCII string should not count itself in its value, only the number of characters that follow. PBS ignores all nulls, tabs, and blanks in an ASCII string, and converts all lowercase characters to uppercase. PBS also ignores a counted ASCII string field with a length byte of 0.

Normally, the individual fields that make up a User Request Packet are stored one after the other in the data buffer. You can leave some room between fields by filling the unused part of the data buffer with nulls, CHR$(0%). In some cases, this may make it easier to deal with variable-length fields.

You can use one of two fields to specify a file:

- The ASCII File Specification Field. This variable-length field contains the name of a RSTS/E file specification as a counted ASCII string.

- The Binary File Specification Field. This fixed-length field contains a RSTS/E file specification in binary format, similar to that returned by the File Name String Scan SYS call (SYS -10).

You must include at least one of these two file specification fields in a User Request Packet. There can be multiple occurrences of either or both file specification fields; all other fields in the packet are optional. The field descriptions that follow discuss the default values assigned to such fields.

Some fields represent file qualifier fields, and are used in conjunction with a file specification field to provide additional information about the file. For example, the /COPIES field indicates the number of file copies to print.

Like file specification fields, you can specify more than one file qualifier field. Generally, you place each file qualifier field right after its file specification field. However, you can specify a file qualifier field only once and have it apply to all file specification fields in the request. PBS interprets file qualifier fields according to these rules:

- If a file qualifier field appears after a file specification field in the data buffer, then it applies only to the last file specification field that precedes it.

- If a file qualifier field precedes all file specification fields in the data buffer, then it applies to all file specification fields in the buffer. This action corresponds to the standard DCL rules for file qualifiers. For example, if a /COPIES field with the value 2 precedes all file specification fields, then PBS prints two copies of each file in the packet. You can override this global file qualifier for a single file specification field by placing a second file qualifier field after it.

- If a file qualifier field does not appear anywhere in a request packet, then PBS assigns a default value. For example, if a data buffer includes no /COPIES field, then PBS prints one copy of each file in the request.

The following fields represent file qualifier fields:

- /[NO]CONVERT Flag field

- /COPIES field

- /[NO]DELETE Flag field

- /[NO]FEED Flag field

- /[NO]FLAG_PAGES Flag field
- /[NO]TRUNCATE Flag field

Only file specification fields and file qualifier fields can occur more than once in a user request packet. If PBS encounters any other field more than once, it uses only the value of the last copy of a duplicate field.

The data buffer that you pass with a user request can be up to 512 bytes long. PBS converts a valid user request packet into a queue entry packet and places the entry in the queue file. The maximum length of an entry packet is also 512 bytes. It is possible for a user request to generate an entry packet that is larger than 512 bytes. In such cases, PBS rejects the user request packet. The following fields can affect the size of the internal entry packet:

- ASCII file specification field
- Binary file specification field
- /PARAMETERS field

Generally, you can prevent an overflow condition by reducing the number of file specification fields or reducing the length of the /PARAMETERS field in a User Request Packet.

Do not include duplicate /PARAMETERS fields in a request, because PBS attempts to allocate space for each occurrence, even though it only uses the rightmost occurrence of the field. As a result, an overflow condition can occur.

Each field described in this section corresponds to a command qualifier, file qualifier, or command parameter in the DCL PRINT and SUBMIT commands. See the *RSTS/E System Manager's Guide* for a complet description of the rules governing the use of the fields.

Table 9-1 summarizes the user request data fields, in ascending order of code number. The remainder of this section describes each data field in detail.

Table 9-1: User Request Data Fields

Code	Field	Request Type
1	/NAME	Any
2	/QUEUE	Any
3	/OWNER	Any
4	/PRIORITY	Any
5	/JOB_COUNT	Print
6	/FORMS	Print
7	/AFTER	Any
8	/PAGE_LIMIT	Print
9	/CPU_LIMIT	Batch
10	/TIME_LIMIT	Batch
11	/PARAMETERS	Batch
12	/[NO]HOLD	Any
13	/[NO]LOG_FILE Flag	Batch
14	/LOG_FILE File Specification	Batch
15	/[NO]LOG_QUEUE Flag	Batch
16	/LOG_QUEUE Name	Batch
17	/LOG_DELETE Flag	Batch
128	ASCII File Specification	Any
129	Binary File Specification	Any
130	/[NO]CONVERT Flag	Print
131	/COPIES	Print

Table 9-1: User Request Data Fields (Cont.)

Code	Field	Request Type
132	/[NO]DELETE Flag	Any
133	/[NO]FEED Flag	Print
134	/[NO]FLAG_PAGES Flag	Print
135	/[NO]TRUNCATE Flag	Print

/NAME Field

This field specifies the name of the entry to be placed in the queue, and corresponds to the entry name specified in the PRINT and SUBMIT commands. If you omit this field or specify a length of 0, PBS uses the file name of the first file specification field that represents a valid entry name. Since entry names cannot consist exclusively of numeric characters, PBS uses the name PRINT or BATCH if it encounters all numeric file names in the request. PBS truncates all names to nine characters.

The format of the /NAME field is:

Byte	Specification
1	CHR$(1%)
2	CHR$(N%), where N% is the length of the name string.
3+	N$, the entry name.

/QUEUE Field

This field specifies the name of the queue on which PBS is to place the request, and corresponds to the queue name specified in a PRINT and SUBMIT commands. If you omit this field or specify a length of 0, PBS places the request on the default queue of the same type (print or batch).

The format of the /QUEUE field is:

Byte	Specification
1	CHR$(2%)
2	CHR$(N%), where N% is the length of the name string.
3+	N$, the queue name.

/OWNER Field

This field specifies the PPN for the owner of the request, and corresponds to the /OWNER qualifier of the PRINT and SUBMIT commands. If you omit this field, the packet sender becomes the owner by default. If the PPN specified is [0,0] or is the same as the sender's, PBS ignores this field.

You need the GACNT privilege to specify a different PPN within the group, and WACNT privilege to specify any PPN. Note that when you specify /OWNER with a PPN other than your own, PBS bases all access rights and privileges for the request on the authorized privileges of that owner's account.

The format of the /OWNER field is:

Byte	Specification
1	CHR$(3%)
2	CHR$(0%)
3	Programmer number.
4	Project number.

/PRIORITY Field

This field defines the priority of the request, and corresponds to the /PRIORITY qualifier of the PRINT and SUBMIT commands. If you omit this field or specify a value of 0, PBS uses the default priority for the queue on which the request is placed. Unless the caller has EXQTA privilege, the priority value cannot exceed the maximum priority defined for the request's queue.

The format of the /PRIORITY field is:

Byte	Specification
1	CHR$(4%)
2	CHR$(N%), where N% is the priority.

/JOB_COUNT Field

This field applies to print requests only. It specifies the number of job copies to be printed, and corresponds to the /JOB_COUNT qualifier of the PRINT command. If you omit this field or specify the value 0, PBS assumes a job count of 1. PBS permits any value in the range 1 to 255.

The format of the /JOB_COUNT field is:

Byte	Specification
1	CHR$(5%)
2	CHR$(N%), where N% is the job count.

/FORMS Field

This field applies to print requests only. It identifies the forms required for the print job, and corresponds to the /FORMS qualifier of the PRINT command. If you omit this field or specify a length of 0, PBS uses the default form name for the queue on which the request is placed. PBS truncates all names to nine characters.

Note that PBS does not verify that the specified form name exists in the Forms Definition file.

The format of the /FORMS field is:

Byte	Specification
1	CHR$(6%)
2	CHR$(N%), where N% is the length of the form name string.
3+	N$, the form name.

/AFTER Field

This field specifies a date and time before which PBS will not process the request, and corresponds to the /AFTER qualifier of the PRINT and SUBMIT commands. If you omit this field, PBS processes the request as soon as possible. The /AFTER Date/Time field consists of a standard RSTS/E date word and a standard RSTS/E time word. If the date word is 0, PBS uses the current system date. If the time word is 0, PBS uses the time 11:59 PM (end-of-day). An error results if the values you specify do not represent a valid date or time.

The format of the /AFTER field is:

Byte	Specification
1	CHR$(7%)
2	CHR$(0%)
3-4	CHR$(D%) + CHR$(SWAP%(D%)), where D% is the after date word.
5-6	CHR$(T%) + CHR$(SWAP%(T%)), where T% is the after time word.

/PAGE_LIMIT Field

This field applies to print requests only. It defines the maximum number of pages that PBS prints in the requested print job, and corresponds to the /PAGE_LIMIT qualifier of the PRINT command. If you omit this field or specify a value of 0, PBS uses the default page limit of the request's queue. Unless the caller has EXQTA privilege, an error results if this value exceeds the maximum page limit of the request's queue.

The double-word value -1 instructs PBS to impose no page limit on the requested print job, and is similar to the /PAGE_LIMIT=UNLIMITED qualifier. An error results if the request's queue does not have its maximum page limit set to UNLIMITED.

The format of the /PAGE_LIMIT field is:

Byte	Specification
1	CHR$(8%)
2	CHR$(0%)
3-4	CHR$(P%) + CHR$(SWAP%(P%)), where P% is the least significant word of the page limit.
5-6	CHR$(P%) + CHR$(SWAP%(P%)), where P% is the most significant word of the page limit.

/CPU_LIMIT Field

This field applies to batch requests only. It defines a CPU limit for the requested batch job, and corresponds to the /CPU_LIMIT qualifier of the SUBMIT command. If you omit this field or specify a value of 0, PBS uses the default CPU limit of the request's queue. Unless the caller has EXQTA privilege, an error results if this value exceeds the maximum CPU limit of the request's queue.

The double-word value -1 instructs PBS to impose no CPU limit on the requested batch job, and is similar to the /CPU_LIMIT=UNLIMITED qualifier. An error results if the request's queue does not have its maximum CPU limit set to UNLIMITED.

The format of the /CPU_LIMIT field is:

Byte	Specification
1	CHR$(9%)
2	CHR$(0%)
3-4	CHR$(C%) + CHR$(SWAP%(C%)), where C% is the CPU limit.

/TIME_LIMIT Field

This field applies to batch requests only. It defines an elapsed time limit for the requested job, and corresponds to the /TIME_LIMIT qualifier of the SUBMIT command. If you omit this field or specify a value of 0, PBS uses the default time limit of the request's queue. Unless the caller has EXQTA privilege, an error results if this value exceeds the maximum time limit defined for the request's queue.

The value -1 instructs PBS to impose no time limit on the requested batch job, and is similar to the /TIME_LIMIT=UNLIMITED qualifier. An error results if the request's queue does not have its maximum time limit set to UNLIMITED.

The format of the /TIME_LIMIT field is:

Byte	Specification
1	CHR$(10%)
2	CHR$(0%)
3-4	CHR$(T%) + CHR$(SWAP%(T%)), where T% is the time limit.

/PARAMETERS Field

This field applies to batch requests only. It contains a string of parameters to be passed to the batch job when it starts, and corresponds to the /PARAMETERS qualifier of the SUBMIT command. You can specify up to eight parameters, separated by one or more spaces or tabs (not commas). PBS accepts only printable characters in the /PARAMETERS string; PBS strips all nonprintable characters from the string. Also, you must place quotes around any individual parameter that includes embedded spaces or tabs. If you omit this field or specify a length of 0, PBS passes no parameters to the batch job.

The format of the /PARAMETERS field is:

Byte	Specification
1	CHR$(11%)
2	CHR$(N%), where N% is the length of the parameter string.
3+	P$, the parameter string.

/[NO]HOLD Flag Field

This field consists of a flag byte that determines whether PBS initially puts the request on hold. PBS does not process an entry on hold until you release it with the SET ENTRY/RELEASE command. This field corresponds to the /[NO]HOLD qualifier of the PRINT and SUBMIT commands.

If you omit this field or specify a value of 0, PBS does not put the request on hold. If the flag byte is nonzero, PBS puts the request on hold.

The format of the /[NO]HOLD field is:

Byte	Specification
1	CHR$(12%)
2	CHR$(N%), where N% is the flag byte. Values are:
	0% No hold (the default)
	1% Hold

/[NO]LOG_FILE Flag Field

This field applies to batch requests only. It consists of a flag byte that determines whether PBS creates a log file for the batch job, and corresponds to the /[NO]LOG_FILE qualifier of the SUBMIT command.

Generally, you include this field to disable logging, since, by default, PBS always creates a log file. If you omit this field or if the flag byte is nonzero, PBS creates a log file at the start of the batch job. If you also include a /LOG_FILE File Specification field, PBS uses that file specification as the log file name.

If the flag byte is 0 and you include a /LOG_FILE File Specification field, PBS use the rightmost rule to resolve the conflicting fields. That is, PBS ignores the first field and determines whether to create a log file based only on the second field.

The format of the /[NO]LOG_FILE Flag field is:

Byte	Specification
1	CHR$(13%)
2	CHR$(N%), where N% is the flag byte. Values are:
	0% No log file
	1% Log file (the default)

/LOG_FILE File Specification Field

This field applies to batch requests only. It specifies the log file that PBS uses for batch processing, and corresponds to the /LOG_FILE=file-spec qualifier of the SUBMIT command.

If you omit this field or specify a length of 0 (and you do not specify a value of 0 in the LOG_FILE Flag field), PBS creates a default log file specification. Generally, you use this field to specify a log file other than the default, since the /LOG_FILE Flag field creates the default log file-spec.

If you specify this field and also specify a value of 0 in the /LOG_FILE Flag field, PBS use the rightmost rule to resolve the conflicting fields. That is, PBS ignores the first field and determines whether to create a log file based only on the second field.

This field contains a single RSTS/E file specification (dev:[PPN]filnam.typ) as a counted ASCII string. If you do not specify a device, PBS assumes _SY:. If you omit the PPN, PBS assumes the sender's PPN. You must specify a file name. If you omit the file type, then PBS appends the file type .LOG.

The format of the /LOG_FILE field is:

Byte	Specification
1	CHR$(14%)
2	CHR$(N%), where N% is the length of the log file specification.
3+	L$, the log file specification.

/[NO]LOG_QUEUE Flag Field

This field applies to batch requests only. It consists of a flag byte that determines whether PBS queues the batch job's log file for printing when the job completes, and corresponds to the /[NO]LOG_QUEUE qualifier of the SUBMIT command.

If you omit this field or specify a flag byte o 0, PBS does not queue for printing any log file created for the batch job. If the flag byte is nonzero, PBS queues the log file for printing on the default queue when the batch job completes. If you also include a /LOG_QUEUE Name field, PBS uses that file specification as the queue name.

The /LOG_QUEUE Name field also affects whether PBS queues any log file for printing. If you specify a flag byte value of 0 and you include a /LOG_QUEUE Name field, PBS use the rightmost rule to resolve the conflicting fields. That is, PBS ignores the first field and determines whether to create a log file based only on the second field.

PBS ignores this field if you disable logging by using a /[NO]LOG_FILE Flag field with a 0 value in its flag byte.

The format of the /LOG_QUEUE Flag field is:

Byte	Specification
1	CHR$(15%)
2	CHR$(N%), where N% is the flag byte. Values are: 0% No log queue (the default) 1% Log queue

/LOG_QUEUE Name Field

This field applies to batch requests only. It defines the queue on which PBS places the log file print request after it completes the batch job, and corresponds to the /LOG_QUEUE=queue-name qualifier of the SUBMIT command.

PBS ignores this field if you disable logging by using a /[NO]LOG_FILE Flag field with a 0 value in its flag byte.

If you omit this field or specify a length of 0 (and you also specify a nonzero value in the /LOG_QUEUE field), PBS places the log file print request on the default print queue.

Generally, you include this field to specify a queue other than the default print queue for PBS to place the log file print request. If you want to place the request on the default print queue, use the /LOG_QUEUE Flag field instead.

The /LOG_QUEUE Flag field also affects whether PBS queues any log file for printing. If you omit the /LOG_QUEUE Flag field or specify a flag value of 0, and you include a /LOG_QUEUE Name field, PBS use the rightmost rule to resolve the conflicting fields. That is, PBS ignores the first field and determines whether to create a log file based only on the second field.

The format of the /LOG_QUEUE Name field is:

Byte	Specification
1	CHR$(16%)
2	CHR$(N%), where N% is the length of the log file queue name.
3+	L$, the log file queue name.

/[NO]LOG_DELETE Flag Field

This field applies to batch requests only. It consists of a flag byte that determines whether PBS deletes the log file after it is printed, and corresponds to the /[NO]LOG_DELETE qualifier of the SUBMIT command.

PBS ignores this field unless you queue a log file for printing by using the /[NO]LOG_QUEUE Flag field and /LOG_QUEUE Name field.

If you omit this field or specify a flag byte of 0, PBS does not delete the log file after printing. If the flag byte is nonzero, PBS deletes the log file after printing.

The format of the /[NO]LOG_DELETE Flag field is:

Byte	Specification
1	CHR$(17%)
2	CHR$(N%), where N% is the flag byte. Values are:
	0% No log delete (the default)
	1% Log delete

ASCII File Specification Field

This field contains the name of a file to be printed (print requests) or a command file to be processed (batch requests), and corresponds to the file specification parameter of the PRINT and SUBMIT commands.

The file specification field contains a single RSTS/E file specification (dev:[PPN]filnam.typ) as a counted ASCII string. If you do not specify a device, PBS assumes _SY:. If you omit the PPN, PBS assumes the caller's PPN. You must specify a file name. If you omit the file type, then PBS appends the .LST file type for print requests or the .COM file type for batch requests. The PPN, file name, and file type fields can contain wildcard characters.

You must define at least one ASCII or Binary File Specification field in a user request packet. One request packet can contain several of these fields.

Note that PBS does not verify the existence of any files matching the field's file specification.

The format of the ASCII File Specification field is:

Byte	Specification
1	CHR$(128%)
2	CHR$(N%), where N% is the length of the ASCII file specification.
3+	F$, ASCII file specification.

Binary File Specification Field

This field contains, in binary format, the file specification to be printed (print requests) or the command file to be processed (batch requests), and corresponds to the file specification parameter of the PRINT and SUBMIT commands.

Unlike the ASCII File Specification field, this field is fixed length. Its format corresponds closely to that of the data string returned by the File Name String Scan SYS call (SYS -10) for a file specification. In most cases, your program can simply copy fields from the data string to the Binary File Specification field in the packet buffer.

The Binary File Specification field contains a single RSTS/E file specification (dev:[PPN]filnam.typ). If the device part is null, PBS assumes _SY:. If the PPN part is null, PBS assumes the caller's PPN. The file name cannot be null. The PPN, file name, and file type fields can contain wildcard characters.

You must define at least one ASCII or Binary File Specification field in a user request packet. One request packet can contain several of these fields.

Note that PBS does not verify the existence of any files matching the field's file specification.

The format of the binary file specification field is:

Byte	Specification
1	CHR$(129%)
2	CHR$(0%)
3	Programmer number.
4	Project number.
5-8	File name in Radix-50 format.
9-10	File type in Radix-50 format.
11-12	Disk device name.
13	Device unit number.
14	Unit number real flag.

/[NO]CONVERT Flag Field

This field applies to print requests only. It consists of a flag byte that determines whether PBS converts all 0 (zero) characters to O ("oh") characters, and corresponds to the /[NO]CONVERT qualifier of the PRINT command.

If the flag byte is nonzero, PBS performs zero-to-oh conversion. If you omit this field or specify a flag byte of 0, PBS does not perform a conversion.

The format of the /[NO]CONVERT Flag field is:

Byte	Specification
1	CHR$(130%)
2	CHR$(N%), where N% is the flag byte. Values are:
	0% Do not convert (the default)
	1% Convert

/COPIES Field

This field applies to print requests only. It indicates the number of file copies to print, and corresponds to the /COPIES file qualifier of the PRINT command.

If you omit this field or specify a value of 0, PBS prints one copy of each file. PBS accepts any value in the range 1 to 255.

The format of the /COPIES field is:

Byte	Specification
1	CHR$(13%)
2	CHR$(N%), where N% is the number of copies.

/[NO]DELETE Flag Field

This field consists of a flag byte that determines whether PBS deletes the file after printing or execution, and corresponds to the /DELETE qualifier of the PRINT and SUBMIT commands.

If the flag byte is nonzero, PBS deletes the file. If you omit this field or specify a flag byte of 0, PBS does not delete the file.

The format of the /[NO]DELETE Flag field is:

Byte	Specification
1	CHR$(132%)
2	CHR$(N%), where N% is the flag byte. Values are:
	0% Do not delete (the default)
	1% Delete

/[NO]FEED Flag Field

This field applies to print requests only. It consists of a flag byte that determines whether PBS performs a form-feed whenever printing reaches six lines from the bottom of the page, and corresponds to the /FEED file qualifier of the PRINT command.

If the flag byte is nonzero or you omit this field, PBS performs the form-feed. If the flag byte is 0, PBS performs no form-feed.

The format of the /[NO]FEED Flag field is:

Byte	Specification
1	CHR$(133%)
2	CHR$(N%), where N% is the flag byte. Values are:
	0% No feed
	1% Feed (the default)

/[NO]FLAG_PAGES Flag Field

This field applies to print requests only. It consists of a flag byte that determines whether PBS prints flag pages at the beginning of each file listing, and corresponds to the /[NO]FLAG_PAGES file qualifier of the PRINT command.

If the flag byte is nonzero, or you omit this field, PBS prints flag pages based on the setting of the request's form. If the flag byte is 0, PBS prints no flag pages.

The format of the /[NO]FLAG_PAGES Flag field is:

Byte	Specification
1	CHR$(134%)
2	CHR$(N%), where N% is the flag byte. Values are:
	0% No flag pages
	1% Flag pages (the default)

/[NO]TRUNCATE Flag Field

This field applies to print requests only. It consists of a flag byte that determines whether PBS truncates lines that exceed the width of the request's form, and corresponds to the /[NO]TRUNCATE file qualifier of the PRINT command.

If the flag byte is nonzero, PBS truncates lines. If the flag byte is 0 or you omit this field, PBS does not truncate lines.

The format of the /[NO]TRUNCATE Flag field is:

Byte	Specification
1	CHR$(135%)
2	CHR$(N%), where N% is the flag byte. Values are:
	0% No truncate (the default)
	1% Truncate

Chapter 10

System Programming Hints

This chapter provides information for designing a BASIC-PLUS program to run by a CCL command. It also describes how the monitor handles the SLEEP and conditional SLEEP statements.

Designing a Program to Run by a CCL Command

Many RSTS/E system programs can be run by special commands called Concise Command Language (CCL) commands. For example, the standard CCL command PIP runs the PIP system program.

CCL commands let you run programs using commands similar to keyboard monitor commands. When you enter a CCL command, the monitor loads and runs a program from a predefined account and device.

CCL commands can run user programs as well as system programs. The system manager can tailor CCL commands to a system by using the DCL DEFINE/COMMAND command (see the *RSTS/E System Manager's Guide*).

CCL commands do not have permanent definitions. Instead, all CCL commands (including those that run DIGITAL programs) must be either installed at system start-up or defined during timesharing with the DEFINE/COMMAND command. See the *RSTS/E System Manager's Guide* for more information.

The rest of this section describes the CCL facility in detail and explains how it interacts with the BASIC-PLUS run-time system. With this information, you can design BASIC-PLUS programs to run by CCL commands.

System Processing of CCL Commands

After you enter a line at your terminal, the run-time system passes the line you entered to the CCL parser in the monitor to see if it is

a valid CCL command. If the line is not a valid CCL command, the monitor returns control to the run-time system. If the line is a valid CCL command, the monitor:

1. Sets up the job's core common area.

2. Extracts the CCL's parameter word from predefined CCL data, which resides in a linked list of monitor buffers. (The BASIC-PLUS run-time system uses the parameter word as the line number where execution is to start; other run-time systems may interpret it differently.)

3. Sets up the program to run.

4. Transfers control to the run-time system associated with the program. The run-time system runs the program.

After the program is finished running, the system returns control to the keyboard monitor that you are working in.

CCL Precedence Rules

In BASIC-PLUS, the system processes CCL commands before BASIC-PLUS keyboard monitor commands and immediate mode statements, but after line-numbered statements. BASIC-PLUS scans terminal input at command level and applies the following rules:

- If the line begins with numbers, the system passes the line to the BASIC-PLUS syntax analyzer for processing and storage as intermediate code.

- If the line begins with nonnumeric characters, the system passes the line to the CCL command parser for processing and validation.

- If the line does not contain a valid CCL command, the CCL parser passes it back to the BASIC-PLUS syntax analyzer for immediate mode execution.

- If the line does not contain a valid command or immediate mode statement, BASIC-PLUS generates an error message.

Thus, a CCL command that duplicates either a BASIC-PLUS command or immediate mode statement overrides that command or statement.

Except for DCL, other standard RSTS/E run-time systems follow the same precedence rules for CCL commands as BASIC-PLUS. Like BASIC-PLUS, these run-time systems process CCL commands before keyboard monitor commands. On the other hand, DCL processes DCL keyboard monitor

commands before CCL commands unless you use the CCL prefix. See the *RSTS/E System User's Guide* for more information.

Effect of CCLs on Your Job Area

Some CCL commands perform the same functions as BASIC-PLUS keyboard monitor commands. Unlike keyboard monitor commands, however, CCL commands destroy the current contents of your job area.

For example, the BASIC-PLUS CATALOG command and the CCL DIR command both display directory listings. When you enter the CATALOG command, BASIC-PLUS calls monitor code to produce the listing. Your job space is not affected. The CCL DIR command, however, loads and runs the DIRECT program from the system library. DIRECT overwrites your job space.

CCL Syntax and Switches

The following lines show the proper syntax for a valid CCL command called COMMAND that can be abbreviated COM. In these lines, <anything> represents characters that the CCL parser does not process.

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

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

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 command line that the run-time system passes to the parser can contain two switches (both optional) in the following format:

```
[<space>]/SI[Z[E]]:[+][#]<digits>[.]
```

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 sign, the digits value is the total size in K words that the program should expand to.
#	indicates the digits value is given in octal. The default is decimal.
<digits>	is the value for size, in K words. Size can be neither less than 1 nor greater than 32 (decimal).
.	explicitly indicates a decimal value for digits.

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

where:

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

The parser strips the optional switches from the command line. The monitor extracts these switches and sets status bits for the run-time system, but 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; however, they must immediately follow the CCL command. If the parser detects a syntax error, it generates either the error ?Illegal switch usage (ERR=67) or the error ?Illegal number (ERR=52). If the command line exceeds 127 characters (the maximum size of core common), the parser generates the error ?Line too long (ERR=47).

Because the parser searches the typed line for special switches, you should not define program switches that 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 that does not begin with a slash (/) character (for example, SY: or another device specification).

CCL Command Line Parsing

When the CCL parser receives a command string, it:

1. Translates the string.
2. Checks the string for a valid CCL command.
3. Writes the fully expanded CCL command into core common, and makes sure that it is delimited by a space.

4. Checks the remaining string for both of the valid CCL switches.

5. Writes the remaining line (except for 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).

Run-time system actions are independent of what the CCL parser does.

To translate the command line, the parser performs several steps:

1. For all characters in the input string, it 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 inside quotation marks, the parser changes all tab characters to spaces, reduces adjacent spaces to a single space, discards all control characters, and converts lowercase letters (CHR$(97) to CHR$(122)) to uppercase letters (CHR$(65) to CHR$(90)).

3. The parser does not alter characters inside quotation marks.

Next, the parser scans the leftmost part of the translated command line for a potential CCL command. The scan ends on the first occurrence of one of the following:

- An end-of-line
- A slash (/)
- A space

Note that if the command begins with a nonalphanumeric character (that is, if the command is a single-character CCL), the scan ends on the second character.

The parser compares the scanned string 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 are critical. A space typed within the CCL command ends 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 valid 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 is not matched with a command whose minimum abbreviation is COM.

Note that in the case of a single-character, nonalphanumeric CCL, you do not need to type the delimiter (such as a space) normally required to set off the command. For example, $ is permanently defined (by the monitor) as the CCL to invoke a DCL command, so a command typed as $COPY is interpreted and passed in core common as $ COPY.

Because of the way the parser interprets CCL commands, the system manager should make sure that similar commands are defined in the correct order. For example, MACRO must be defined before MAC during system start-up. See the *RSTS/E System Manager's Guide* for more information.

When the parser determines 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 found later by the parser result in unpredictable contents in the core common area.

If the rest of the translated string begins with either a slash or a space followed by a slash, the parser checks for a valid CCL switch. If it finds a valid CCL switch, the parser checks further for another adjacent switch. Duplication of a switch generates the error ?Illegal switch usage (ERR=67). Any CCL switches found are removed 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. The monitor passes a flag from the CCL command definition to the run-time system. The monitor passes the fully defined CCL command and the remaining string in the core common area.

BASIC-PLUS Action

The BASIC-PLUS run-time system 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 inline. When the monitor fails to validate a CCL command, BASIC-PLUS processes the translated string for execution in immediate mode. If the parser returns an error, BASIC-PLUS prints the error 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:

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 these actions, BASIC-PLUS runs the program.

Table 10-1 gives the rules that BASIC-PLUS uses when setting the STATUS variable.

Table 10-1: 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 for a CCL entry).

If BASIC-PLUS finds that the line number is nonzero, it checks the flag passed by the monitor. BASIC-PLUS permanently drops temporary privileges unless the CCL definition indicates 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 /DET switch was specified, BASIC-PLUS detaches the job and closes all channels on which the

console terminal is open.

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. Other 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.

Conventions Used in 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 transfer control to other routines in the program. This convention allows programs to determine 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. After determining which CCL command caused it to run, the program can transfer control to routines to process the rest of the command line.

SLEEP and Conditional SLEEP Statements

The BASIC-PLUS SLEEP statement lets a running program stop its own execution for a specified time period. The statement has the format:

```
SLEEP <expression>
```

The <expression> indicates the number of seconds to stop execution. The system suspends the job that is controlling the program, and execution stops until the system "awakens" the job at the end of the specified time period. See the *BASIC-PLUS Language Manual* for more information.

Although the program controls the sleep state by specifying the time period for the system to stop execution, certain conditions cause the system to awaken the job before the time is up:

- A user enters a delimiter (RETURN, LINE FEED, FORM FEED, or ESCAPE) at:
  - Any terminal opened by the job.
  - Any terminal allocated to the job if the job also has a keyboard open on a nonzero channel.

- A dial-up line that is allocated or opened by the job gets hung up.
- The system manager disables logins (that is, sets the number of logins to 1).
- A state change occurs on a pseudo keyboard opened by the job. This condition can occur when the opened pseudo keyboard has output for the controlling job or has entered an input wait state. See the section "Pseudo Keyboards" in Chapter 4 for more information.
- The job has declared itself a receiver and a message is queued for it through the Send/Receive SYS calls (see Chapter 8).
- The job has a DMC/DMR (XM:) device open and the device driver receives a message (see Chapter 6).
- The system date or time is changed.

You can specify a conditional SLEEP by setting the sign bit in the SLEEP statement argument. To set the sign bit, specify the SLEEP statement argument as:

```
SLEEP <expression> + 32767% + 1%
```

In a conditional SLEEP, the monitor checks for all conditions except disabled logins before executing the SLEEP statement. The monitor does not execute the SLEEP statement if the job:

- Has pending terminal input
- Has a dial-up line allocated but not in use
- Has a message queued through send/receive
- Has data pending on the DMC/DMR device (XM:)
- Has a pseudo keyboard open that has pending output or is waiting for input

If any conditions (other than NO LOGINS) that cause a job to to be awakened is in effect when the conditional sleep is issued, the sleep does not take place. A "normal" sleep does not check these conditions before sleeping, so the job is awakened only when one of these conditions occurs again, or when the timer expires.

Chapter 11

Ethernet Operations

This chapter presents an overview of Ethernet and describes how to use its local area networking features on RSTS/E with BASIC-PLUS or BASIC-PLUS-2. Some special functions work only through MACRO-11 programs. You can use Ethernet without these functions, but they can greatly increase an application's flexibility and its ability to monitor the network. For descriptions of these special functions, see the *RSTS/E System Directives Manual*, under the .SPEC listing for Ethernet.

Ethernet Concepts

Ethernet consists of a single coaxial cable that connects computers, terminals, and other devices within a limited geographic area. All nodes on an Ethernet have equal access to the interconnecting cable. Ethernet's access method is called "Carrier-Sense, Multiple-Access with Collision Detect" (CSMA/CD). These terms mean:

- Carrier Sense -- Each node checks the cable before it sends a message or data packet. If another node is transmitting, the first node delays transmission until the cable is no longer busy.

- Multiple Access -- All nodes are on the same coaxial cable, and all the nodes can hear all message or data packets sent on the Ethernet. The intended recipient nodes recognize incoming packets by addresses which are specified within the packets.

- Collision Detect -- If two or more nodes send packets at the same time, their signals collide. Each node hears such collisions, then waits before sending a packet again.

Ethernet Operations

To use Ethernet, you only need four BASIC statements: OPEN, CLOSE, GET, and PUT. See the section Commands for Ethernet for full descriptions of these functions. You may also wish to use the special Ethernet .SPEC functions that have been added to MACRO-11. See the *RSTS/E System Directives Manual* for full descriptions.

The Conversation Analogy

In many ways, Ethernet resembles ordinary conversation at a social gathering. To be polite, you do not speak while someone else is talking; you listen before you speak. This resembles the carrier-sense feature of Ethernet; each node makes sure the cable is clear before sending any information.

In a conversation, anyone may begin to talk once he or she determines that no one else is talking. (Compare this to a lecture, where only one person talks.) This equal right to speak resembles the multiple-access feature of Ethernet; many nodes can use the same cable.

If two people start to talk at the same time, they note the fact and stop talking (that is, each listens while talking and stops if interfering with someone else). This resembles the collision-detect feature of Ethernet; if two nodes start transmitting at the same time, both nodes detect this and stop.

When the two people stop talking, they wait and start over again. On Ethernet, this situation is called backoff and retransmission; a delay before retransmission will eventually clear the collision situation.

There is another useful analogy between Ethernet and a social event. When someone at the party talks, everyone (usually) can hear what is being said. Some of what is said is intended for everyone, some is intended for a smaller group (for example, everyone over 21), and some is intended for an individual. Likewise, nodes on an Ethernet can hear every message. Some messages are intended for all nodes (broadcast address), some are intended for a subset (multicast address), and some are intended for individual stations (physical address).

Ethernet and DECnet/E

DECnet/E is a DIGITAL product using the Ethernet data link layer and Ethernet physical link layer to communicate. It uses the Digital Network Architecture for network control. In order to increase the

flexibility of Ethernet on RSTS/E, DIGITAL has provided a direct interface to the Ethernet data link layer for RSTS/E users. This interface resembles the interface provided for the DMC/DMR communications devices, and can be programmed with or without DECnet/E on the system.

If DECnet/E is on the system, you should start it before any other jobs are allowed to perform OPENs to the Ethernet devices.

Ethernet Terms

To make Ethernet easier to use, you should become familiar with these terms:

- physical layer
- channel, controller, and data link layer
- protocol type and portal
- counters
- physical addressing and hardware addressing
- multicast addressing

Physical Layer

Digital Equipment Corporation, Intel Corporation, and Xerox Corporation collaborated in producing the Ethernet specification to develop a variety of local area network products. DIGITAL's implementation of the Ethernet specification consists of the lowest two levels of the overall DNA specification -- the *physical layer* and the *data link layer*.

The physical layer of Ethernet is a bus in the shape of a branching tree. The medium is a shielded coaxial cable using Manchester-encoded, digital signaling. Each Ethernet can support up to 1023 nodes. The maximum length of the cable is 2.8 kilometers (1.74 miles).

Ethernet Operations

Channel, Controller, and Data Link Layer

Each Ethernet has one *channel*. The channel is made up of the physical cable connecting the nodes, together with the nodes' *controllers*. A controller is a RSTS/E device connected directly to the cable. Each node has one or two controllers connecting to the Ethernet. The controllers and their device drivers make up the Ethernet *data link layer*. Controllers come in three types:

- DELUA, a UNIBUS controller (called UNA in DECnet, XE: in RSTS/E)
- DEUNA, an older UNIBUS controller (called UNA in DECnet, XE: in RSTS/E)
- DEQNA, a Q-Bus controller (called QNA in DECnet, XH: in RSTS/E)

Protocol Type and Portal

All incoming Ethernet messages have a *protocol type*, an identifying string near the beginning, that identifies the proper *portal* to receive the message (for instance, the DECnet/E portal). The portal is the logical access from the user software to the channel.

Counters

The Ethernet controller keeps records of link performance called the *counters*. For example, the counters record the number of times the controller had to throw away a packet because it ran out of buffer space. Use the counters to find problems and fine-tune the system. For example, if the counters show the controller throws away packets too often, you should give the controller more buffers, or read from it more often.

There are two kinds of counters, *circuit counters* and *line counters*. A circuit counter monitors a single portal. A line counter monitors the whole channel. You cannot work with counters through BASIC programs; you must use MACRO-11.

Physical Addressing

You address nodes on Ethernet lines by their Ethernet *physical addresses*. Because the Ethernet is a multiaccess broadcast device, all nodes connected to an Ethernet line are equally accessible. Therefore, each node on an Ethernet is assigned a unique Ethernet physical address which is set by the controller software at the node, or is set to a default value at the factory. This default physical address is the *hardware address*. Xerox Corporation assigns a block of hardware addresses for DIGITAL to use with its DEUNA, DELUA, and DEQNA Ethernet controllers. One address from the assigned block is permanently associated with each controller in read only memory.

Ethernet addresses are represented by six pairs of hexadecimal numbers separated by hyphens, AA-01-23-45-D7-0C for example.

DECnet/E on Ethernet

If you have DECnet/E, the controller software sets the physical address to be within an assigned block of addresses when the node is powered up. The controller constructs the physical address by appending a hexadecimal number to the constant hexadecimal number AA-00-04-00. The controller software uses the node address (area-number.node-number) to construct the last two pairs of hexadecimal numbers it appends to the constant, D7-0C for example. In this case, the physical address is AA-00-04-00-D7-0C.

NOTE

> The system manager must start DECnet/E before any user portals open. Once a portal opens, no users can modify the controller's characteristics.

Multicast Addressing

Use multicast address to send messages to more than one node. A multicast address can be:

- A multicast group address, which is an address assigned to any number of nodes. Use the group address to send a message to all nodes in the group with a single transmission.
- The broadcast address, which is a single address, the hexadecimal number FF-FF-FF-FF-FF-FF. Use a broadcast address to transmit a message to all nodes on a given Ethernet.

NOTE

The use of the broadcast address on Ethernet severely burdens the network resources. DIGITAL does not recommend using a broadcast address on a heavily-populated Ethernet.

Ethernet Addresses

Certain Ethernet addresses and ranges of addresses have specialized functions. DIGITAL physical addresses are in the range:

AA-00-00-00-00-00 through AA-00-04-FF-FF-FF

Multicast addresses assigned for use in cross-company communications are:

Value	Meaning
FF-FF-FF-FF-FF-FF	Broadcast
CF-00-00-00-00-00	Loopback assistance

DIGITAL multicast addresses assigned to be received by other DIGITAL nodes on the same Ethernet are:

Value	Meaning
AB-00-00-01-00-00	Dump/load assistance
AB-00-00-02-00-00	Remote console
AB-00-00-03-00-00	All phase IV routers
AB-00-00-04-00-00	All phase IV end nodes
AB-00-00-05-00-00 through AB-00-03-FF-FF-FF	Reserved for future use
AB-00-04-00-00-00 through AB-00-04-FF-FF-FF	For use by DIGITAL customers for their own applications

Commands for Ethernet

You can use the following BASIC statements on the Ethernet:

- OPEN
- CLOSE
- GET
- PUT

In addition, only the Ethernet controllers use the following special functions:

- Set New Physical Address
- Enable Multicast Addresses
- Get Circuit Counters
- Get Line Counters
- Transfer Circuit Counters
- Transfer Line Counters

Programs written in BASIC-PLUS and BASIC-PLUS-2 can use the OPEN, CLOSE, GET, and PUT statements to operate the Ethernet interface. To use the special functions, you must use MACRO-11 programs. See the *RSTS/E System Directives Manual*, the .SPEC listings for Ethernet.

I

mple Open statement:

PEN "XE0:/PO:1600" AS FILE #1, CLUSTERSIZE 4, RECORDSIZE 512%+6%+6%+2%+2%

- **XE0** specifies Ethernet controller 0
- **/PO:1600** specifies protocol type 1600. This is the position modifier.
- **FILE #1** specifies RSTS/E channel 1.
- **CLUSTERSIZE 4** specifies four system receive buffers. You cannot specify more than 127. DIGITAL does not recommend specifying more than 10.
- **RECORDSIZE 512%+6%+6%+2%+2%** specifies 512 bytes for the size of the I/O buffer, with 6 bytes each for the source and destination addresses, 2 bytes each for the portal protocol type and the Ethernet length field.
- **MODE 0%** is the default and so was not written in the example. 0% defines the portal as using a "padded" protocol. Use MODE 128% for an "unpadded" protocol. (See below for descriptions of padded and unpadded protocols.)

format of the OPEN statement for Ethernet is:

N "XEa:/PO:b" AS FILE #c, CLUSTERSIZE d, RECORDSIZE e%+6%+6%+2%+2%, MODE f%

re *a*, *b*, *c*, *d*, *e*, and *f* are the variables described in the ceding example.

the OPEN statement to open a portal on a given Ethernet troller. The OPEN statement also lets you allocate receive buffers set the portal protocol type.

er the OPEN statement, the portal receives incoming messages for specified protocol type at the physical address of the controller.

NOTE

If you intend to use DECnet/E, be sure you start DECnet/E before you issue any OPEN statements for users. Since DECnet/E has to modify the node's physical address before it can start, it must be the first portal opened on a channel.

Possible Errors

Meaning	ERR Value
?NO BUFFER SPACE AVAILABLE There are not enough buffers available in the small buffer pool to create the portal's data structures, or the extended buffer pool (XBUF) is too small or fragmented to allocate the requested number of system receive buffers.	32
?ACCOUNT OR DEVICE IN USE The protocol type requested is already open on the channel.	3
?DEVICE HUNG OR WRITELOCKED The controller is disabled or inoperative.	14

Note the following restrictions:

- Each portal supports only one protocol type. On OPEN, the protocol for the portal is defined. You can enable multiple protocol types by opening several different portals on different RSTS/E channels.
- You cannot open the same protocol type on two portals on the same channel.
- The Ethernet controller physical address is not available to anyone above the data link layer.

Ethernet Operations

Padded and Unpadded Protocols

Protocols may be *padded* or *unpadded*. When you issue an OPEN statement, you must decide whether to do the following send and receive operations in padded or unpadded mode. Specify MODE 0% for a padded protocol, MODE 128% for unpadded. Padded is the default. It is easier to use but takes up more space.

In the padded mode, the data link layer automatically fills in the length field of the receive buffer and makes sure the message is long enough to be put on Ethernet, using the length field of the packet. It also uses the length field of incoming packets. In the unpadded mode, the data link layer leaves the length field blank and leaves it to you to make sure you have the minimum length.

System Receive Buffers

Since a user job may not be in memory when a message for it arrives on the Ethernet, RSTS/E lets you allocate *system receive buffers* to hold messages until the job can pick them up (using GET statements). Allocate these system receive buffers using the CLUSTERSIZE parameter on the OPEN statement.

Under the current version of RSTS/E, each system receive buffer is 632 bytes long. Every message received for the portal uses at least one buffer, and long messages may use as many as three of these system receive buffers, depending on their length.

RSTS/E keeps careful count of available system receive buffers for each portal. When a message comes in for a portal and there are not enough system buffers available to the portal, RSTS/E discards the message. When the user next issues a GET command, it returns ERROR 13 (?Data Error on Device), meaning that at least one message was dropped by RSTS/E due to a shortage of system receive buffers.

On a normal GET command, the system copies a message (of one or more system receive buffers) into the buffers in the user program. Once this copy is complete, the system receive buffers are once again available to receive incoming messages.

DIGITAL does not recommend allocating more than 10 system receive buffers to a portal. DIGITAL also recommends that user portals do not routinely handle messages which are larger than can fit into one system receive buffer.

CLOSE

Example CLOSE statement:

```
CLOSE #1%
```

- **#1%** specifies the device open on RSTS/E channel 1 as the device with the portal to close. The format to close any channel *n* is:

  ```
  CLOSE #n%
  ```

Use the CLOSE statement to close the portal on a given Ethernet controller. RSTS/E closes the portal on the data link side and frees all the system resources reserved for it for other system processes. The CLOSE statement requires no parameters and returns no errors.

GET

Example GET statement:

```
GET #1% &, RECORD 0%
```

- **#1% &** specifies the device on RSTS/E channel 1 as the device with the portal to read from.
- **RECORD 0%** specifies that the GET operation should not be stalled. If there are new messages waiting, the GET operation returns the first one. If not, the GET fails with the error message ERR 5 (?Can't find file or account).

  Users can also use RECORD 8192% to specify a stall for the GET. With a stall, the GET returns the first message, if there are any messages waiting. Otherwise, it stalls the job in an XE state, waiting for an Ethernet message addressed to the portal. Users can interrupt this stalled GET with CTRL/C, in case nothing comes over the Ethernet.

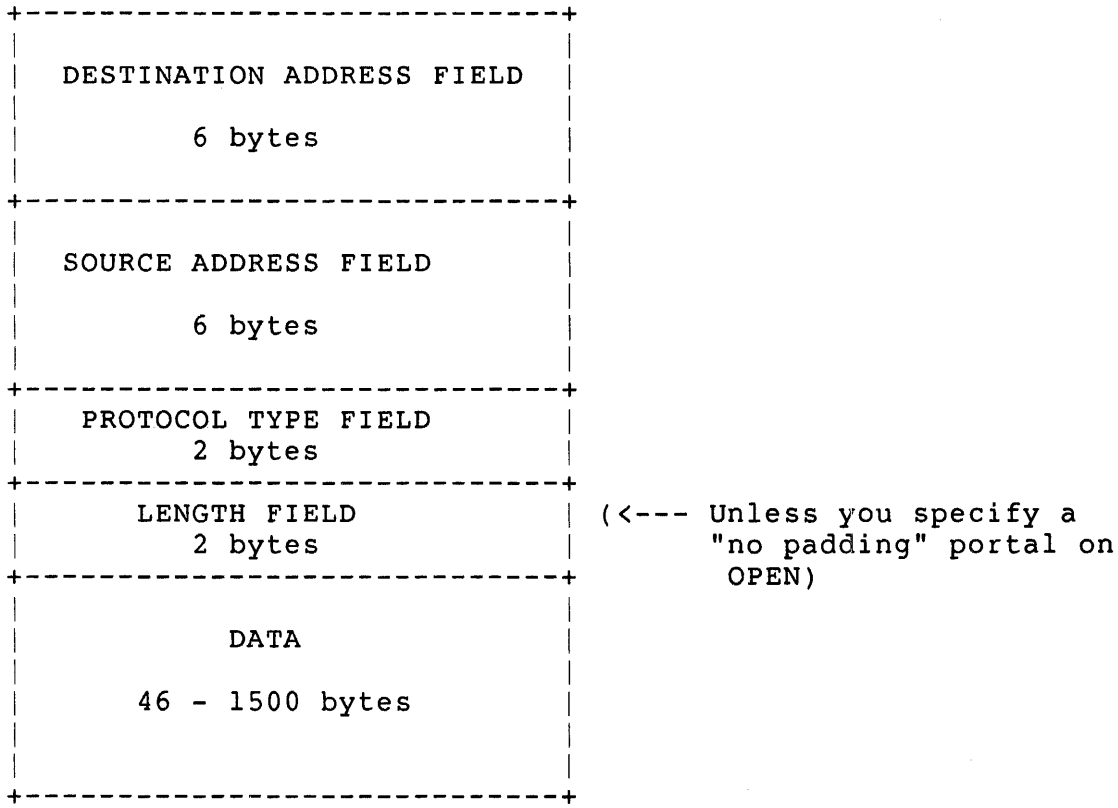
The format for any channel *a*, stalled or not according to the value of *b*, is:

```
GET #a% &, RECORD b%
```

Use the GET statement to read data from a portal previously opened on the channel. If the portal was OPENed in padded mode, then the first 16 bytes are header information, including the length field. If the portal was OPENed in unpadded mode, only the first 14 bytes are header information, followed immediately by message data.

The amount of data read depends on the device and the size of the buffer area, as defined in the XRB. The number of bytes transferred is always less than or equal to the buffer size. The actual number of bytes read is returned in the XRB when the directive is complete.

The receive buffer, which must start on a word boundary, contains the following fields upon completion of a GET:



Possible Errors

Meaning	ERR Value
?MAGTAPE RECORD LENGTH ERROR Message truncated to fit.	40
?DATA ERROR ON DEVICE Lost packets (user buffer unavailable).	13
?CAN'T FIND FILE OR ACCOUNT For no stall GETs, when no messages are pending.	5

PUT

Example PUT statement:

```
PUT #1%, COUNT 6%+6%+2%+2%+LEN(D$)
```

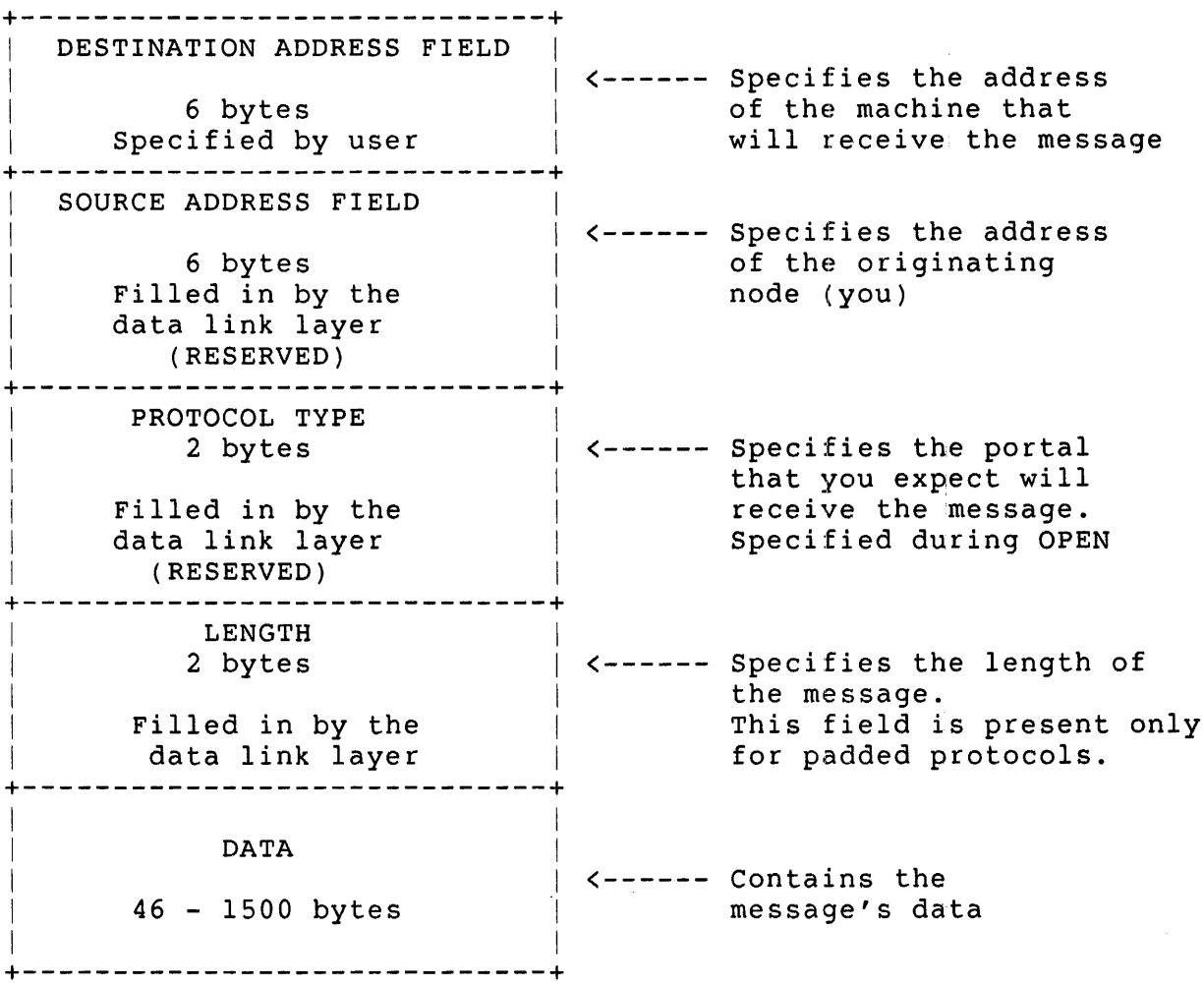
- **#1%** specifies the device on RSTS/E channel 1 as the device with the portal to write to.
- **COUNT 6%+6%+2%+2%+LEN(D$)** specifies 6 bytes each for the source and destination addresses, 2 bytes each for the portal protocol type and the Ethernet length field, and LEN(D$) for the length of the user string to send (that is, the data to be sent is in buffer D$).

The format for any channel *a*, field length *b*, is:

```
PUT #a%, COUNT 6%+6%+2%+2%+b
```

Use the PUT statement to send information through the data link layer to another node. In padded mode, the data link layer fills in the length bytes with the length of the data sent. The data link layer always makes sure the transmit packet is between 60 and 1514 bytes long, and will fill in the source address and protocol type, using the type passed in the OPEN statement.

The PUT statement expects you to PUT a buffer in the following format, providing for space for the following information in addition to the actual data to be transferred:



Note the following requirements and restrictions:

- The total buffer size must be between 60 and 1514 bytes in length
- The buffer must start on a word boundary, but can contain an even or odd number of bytes
- The user specifies the destination address field

- The data link layer always specifies the source address field, which is reserved for DIGITAL use
- The data link layer also specifies the protocol type, which is reserved for DIGITAL use
- If the protocol is a padded protocol, then the data link layer fills in the length field as calculated from the information passed in the XRBC

Possible Errors

Meaning	ERR Value
?ILLEGAL BYTE COUNT FOR I/O The count is not between 60 and 1514 bytes, or starts on an odd address.	31
?DATA ERROR ON DEVICE The device is disabled or inoperative.	13
?DEVICE HUNG OR WRITE LOCKED The controller is disabled or inoperative.	14

Special Ethernet Functions

MACRO-11 provides the following functions to give greater flexibility in using and monitoring the Ethernet. These functions are not available in BASIC. See the *RSTS/E System Directives Manual* under .SPECs for Ethernet for more information.

Set New Physical Address

Use the Set New Physical Address function to change the physical address of the Ethernet controller. It is a .SPEC function and requires too many parameters to call using BASIC-PLUS or BASIC-PLUS-2. Use MACRO-11.

Ethernet Operations

Enable Multicast Addresses

Use the Enable Multicast Addresses function to let the portal receive multicast messages. This is a device dependent .SPEC function. The XRB contains pointers identifying the User Multicast Address Buffer. RSTS/E allows a maximum of five multicast addresses per portal on an Ethernet channel.

This is a .SPEC function and requires too many parameters to call using BASIC-PLUS or BASIC-PLUS-2. Use MACRO-11.

Get Circuit Counters and Get Line Counters

Use the Get Counters functions to bring the counters up to date. The controllers maintain counters in several places. You must tell the data link layer when you want to collect them. The controllers update line or circuit counters only when you issue the call.

These are .SPEC functions and require too many parameters to call using BASIC-PLUS or BASIC-PLUS-2. Use MACRO-11.

Transfer Circuit Counters and Transfer Line Counters

Use the Transfer Counter functions to read the counter information from the data link layer to the user space once you have updated the information with the Get Counters function.

These are .SPEC functions and require too many parameters to call using BASIC-PLUS or BASIC-PLUS-2. Use MACRO-11.

Appendixes

Appendix A

Magnetic Tape Label Formats

RSTS/E supports two magnetic tape file label formats: DOS and ANSI. This appendix discusses DOS and ANSI label formats and describes how RSTS/E handles tapes written with these labels. Note that the ANSI label format described in this appendix refers to the RSTS/E implementation of the American National Standard X3.27-1978 (magnetic tape labels and file structure for information interchange).

This appendix uses the following terms:

Record	A physical record on a magnetic tape. It is the unit of data transferred in a magnetic tape drive operation. Each record on a magnetic tape is separated from the next by a gap or blank space.
Tape Mark	A special kind of record that magnetic tape drives can write and detect. A tape mark contains no data. Instead, it separates other kinds of records.

DOS Magnetic Tape Format

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

The first part of the magnetic tape is a physical beginning-of-tape (BOT), which is a reflective (metallic) marker. Right after this marker is the first tape label record followed, in this case, by ten data records and a tape mark.

All magnetic tape files begin with a tape label record, contain any number of data records (the default size is 512 bytes per record), and end with a tape mark. DOS files can contain zero data records, but a label record and tape mark are always required for each file.

Figure A-1 shows the layout of a DOS magnetic tape file.

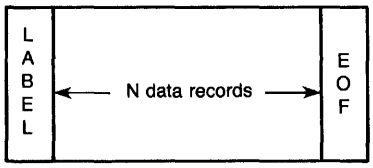


Figure A-1: DOS-Labeled Magnetic Tape File

Figure A-2 shows the layout of a DOS magnetic tape that contains three files of ten data records apiece.

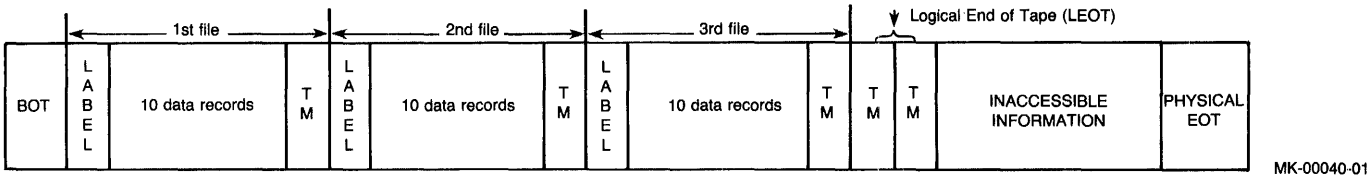


Figure A-2: DOS Magnetic Tape Consisting of 3 Files of 10 Data Records Apiece

After the first file, another label record begins the second file. This label record is also followed by ten data records and a tape mark. This second file is immediately followed by the third and last file, which consists of a label record, ten data records, and a tape mark. In addition, since the third file on this tape is also the last one, two more tape marks follow. The magnetic tape has three tape marks at this point, signifying a logical end-of-tape (LEOT).

After 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 tape marks follow the beginning-of-tape marker.

DOS Labels

The label record that specifies the beginning of a magnetic tape file in DOS format is 14 bytes long. Table A-1 shows the information contained in each of the label record bytes, numbered from 0 to 13.

Table A-1: DOS Label Record Bytes

Byte	Contents	Data Format
0,1,2,3	File name	2 words in RADIX 50.
4,5	File type	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 (decimal)).
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 DOS format magnetic tapes are to be interchanged with DOS-11, RSX-11 or VMS systems, a problem may occur because RSTS/E treats project-programmer numbers as decimal values, and the others treat these numbers (called UICs) as octal values. To avoid interchange problems, simply write all files on the tape with a [1,1] project-programmer number, which is the same in both decimal and octal. For example:

```
100 OPEN "MT0:[1,1]ABC" FOR OUTPUT AS FILE 1%
```

Note that the project-programmer number is part of the file name 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.

ANSI Magnetic Tape Format

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

The first part of the magnetic tape is a reflective physical beginning-of-tape marker. The next item is a volume label (VOL1). (A volume is a reel of magnetic tape. A volume, which may be part of a volume set, can contain part of a file, a complete file, or more than one file.)

The first RSTS/E file begins with two label records, called header labels (HDR1 and HDR2). These header labels are followed by a tape mark. In this case, ten data records are written immediately after the tape mark. The data records are followed, in order, by a tape mark, two trailer label records (EOF1 and EOF2 or EOV1 and EOV2), and another tape mark.

When a file is created but no data blocks are written, all the above label records and tape marks are still present. These labels and end-of-file markers are always required for each file.

Figure A-3 shows the layout of an ANSI magnetic tape file.

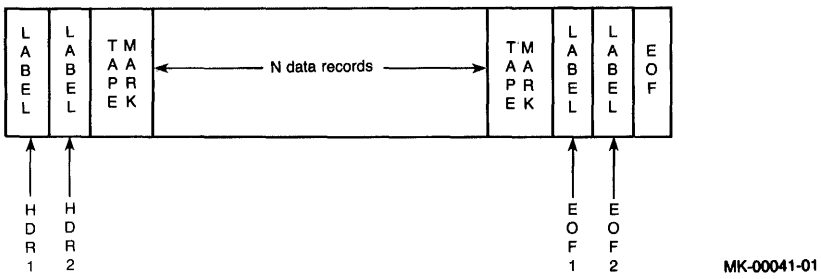


Figure A-3: ANSI-Labeled Magnetic Tape File

Figure A-4 shows the layout of an ANSI magnetic tape that contains three files of ten data records apiece.

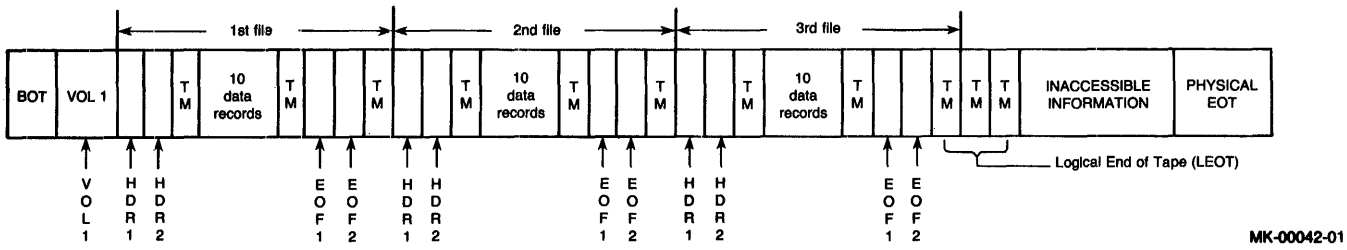


Figure A-4: ANSI Magnetic Tape Consisting of 3 Files of 10 Data Records Apiece

After the first file, another set of two header records begins the second file. The second file is identical to the first one, consisting of the two header labels, one tape mark, ten data records, another tape mark, two trailer labels, and a final tape mark.

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

After 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. When no file exists on the tape (as on an initialized magnetic tape), a dummy file is present with a complete set of labels and tape marks.

ANSI Labels

Each ANSI label record written by RSTS/E is 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.

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)	Space means no restrictions
12-37	Reserved	Spaces
38-51	Owner Identifier* D%B4431JJJGGG	Contents of this field 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 programmer 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.

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.TYP; 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	Numeric characters; starts at 0001. Identifies a section in the file. Specified with the /POSITION switch on the OPEN statement. Defaults to 0001.
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

Table A-3: Header 1 Label Format (Cont.)

Character Position	Field Name and RSTS/E Usage	Contents
55-60	Block Count	000000
61-73	System Code DECRSTS/E	Name of system that produced the volume. Padded by spaces.
74-80	Reserved	Spaces

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.

Table A-4: Header 2 Label Format (Cont.)

Character Position	Field Name and RSTS/E Usage	Contents
16-50	System Dependent (M is the default)	Bytes 16-36 (Spaces) Byte 37 = A means first byte of record contains FORTRAN control character. = (Space) means LF character precedes and CR character follows each record. = M means record contains all form control information.
51-52	Buffer Offset (00)	Not supported by RSTS/E; always 00.
53-80	Reserved	Spaces

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

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. 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. For information on supporting EOV, see the magnetic tape SPEC% function in Chapter 2.

Table A-5 shows the character position, field name, and contents of each byte in the label.

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.TYP; 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 the first volume in the volume set.
28-31	File Section Number	Numeric characters; starts at 0001. Identifies a section in the file. Specified with the /POSITION qualifier on the OPEN statement. Defaults to 0001.
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

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

Character Position	Field Name and RSTS/E Usage	Contents
55-60	Block Count	Total number of blocks in this file section.
61-73	System Code DECRSTS/E	Name of system that produced the volume. Padded by spaces.
74-80	Reserved	Spaces.

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 or EOV
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.

Table A-6: End-of-File or Volume (EOF or EOV) 2 Record Format (Cont.)

Character Position	Field Name and RSTS/E Usage	Contents
16-50	System Dependent (M is the default)	Bytes 16-36 (Spaces) Byte 37 = A means first byte of record contains FORTRAN control character. = (Space) means LF character precedes and CR character follows each record. = M means record contains all form control information.
51-52	Buffer Offset (00)	Not supported by RSTS/E; always 00.
53-80	Reserved	Spaces

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

Initializing Magnetic Tapes

This section describes how RSTS/E initializes (zeros) DOS and ANSI magnetic tapes.

To initialize a magnetic tape written in DOS format, RSTS/E:

1. Rewinds the magnetic tape.
2. Writes three tape marks on the tape.
3. Rewinds the magnetic tape again.

To initialize a magnetic tape written in ANSI format, RSTS/E:

1. Rewinds the magnetic tape.
2. Writes a volume label (VOL1) on the tape. The volume identifier is in bytes 5 through 10, in ASCII.

3. Writes two header labels (HDR1 and HDR2).

4. Writes two tape marks.

5. Writes two trailer labels (EOF1 and EOF2).

6. Writes three tape marks.

7. Rewinds the magnetic tape again.

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

To zero a tape on RSTS/E, use one of the following:

- The Zero a device SYS call (see Chapter 7)
- The UU.ZER directive (see the *RSTS/E System Directives Manual*)
- The DCL INITIALIZE command (see the *RSTS/E System User's Guide*)
- The PIP program (see the *RSTS/E Utilities Reference Manual*)

Appendix B

Card Codes

The RSTS/E card reader driver can be configured for one of four different ASCII punched card codes:

- o ANSI
- o DEC029
- o DEC026
- o 1401

The system manager determines the set of codes used on the system. 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 for DEC029, DEC026, 1401, and the ASCII equivalent.

Table B-1: Card Reader Codes

Character	ASCII10	ANSI and 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

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

Character	ASCII10	ANSI and DEC029	DEC026	1401
$	36	11 8 3	11 8 3	11 8 3
%	37	0 8 4	0 8 7	0 8 4
&	38	12	11 8 7	12
'	39	8 5	8 6	12 8 4
(	40	12 8 5	0 8 4	8 7
)	41	11 8 5	12 8 4	0 8 7
*	42	11 8 4	11 8 4	11 8 4
+	43	12 8 6	12	0 8 5
,	44	0 8 3	0 8 3	0 8 3
-	45	11	11	11
.	46	12 8 3	12 8 3	12 8 3
/	47	0 1	0 1	0 1
0	48	0	0	0
1	49	1	1	1
2	50	2	2	2
3	51	3	3	3
4	52	4	4	4
5	53	5	5	5
6	54	6	6	6
7	55	7	7	7
8	56	8	8	8
9	57	9	9	9
:	58	8 2	11 8 2	8 5

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

Character	ASCII10	ANSI and DEC029	DEC026	1401
;	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

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

Character	ASCII10	ANSI and DEC029	DEC026	1401
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
^	94	11 8 7	8 5	unused
_	95	0 8 5	8 2	12 8 7

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

Appendix C

Error Messages

RSTS/E generates messages 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). The next two sections describe the RSTS/E error messages.

Different messages are generated while a job is executing programs written in languages other than BASIC-PLUS. Such programming languages include COBOL, BASIC-PLUS-2 and FORTRAN-IV. For information about these error messages, consult the appropriate User's Guides. See Table C-6 for a summary of BASIC-PLUS-2 errors.

In most cases, if you are not trapping errors (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. For example:

```
10    OPEN 'Z' FOR INPUT AS FILE 1%
RUNNH
?Can't find file or account at line 10

Ready
```

As the Ready prompt indicates, control returns to the system.

One exception to this procedure occurs when you execute an INPUT statement at the job's console terminal and error trapping is not in effect. The system generates the error message and executes the statement again:

```
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.

Each message has an associated 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 255. An error with a value between 1 and 70 causes the system to transfer control to the line number indicated in the ON ERROR GOTO statement. The system does not print the error message. Your program can check the ERR variable and perform a recovery procedure. If the error value is between 71 and 127, the system does not transfer control to the recovery routine but prints the message and returns control to the system. Error numbers above 127 are reserved for BASIC-PLUS-2. 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 nonrecoverable. The recoverable error messages are listed in ascending order of their error values. A program can use these error values to differentiate errors. Non-recoverable errors are in alphabetical order without error numbers because a program cannot use these numbers 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.
?	Error	Execution cannot continue unless you remove the cause of the error.
??	Severe Error	Execution cannot continue, and you probably cannot remove the cause of the error. In most cases, there is no opportunity for recovery.
none	Information	A message beginning with neither a question mark nor a percent is for information only.

In the error message descriptions in the first two sections, the abbreviations 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, you should document the conditions under which the error occurred and have the appropriate person at your site send an SPR to DIGITAL. Section C.4 contains instructions for filling out an SPR.

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. For example:

```
100  ON ERROR GOTO 0 \ A% = 32768.
200  PRINT A%
RUNNH
%Integer error at line 100
 0

Ready
```

The attempt to compute a value outside the range for integers generates the INTEGER ERROR at line 100. After BASIC-PLUS prints the error message, processing continues but with the conditions described in the error meaning. BASIC-PLUS substitutes 0 for the erroneously computed value.

The number of RSTS/E error messages is restricted to 255. Because of this restriction, certain error messages have multiple meanings. The specific meaning of an error message depends on the operation you are performing when the error condition occurs. For example, if the system attempts a file access and the file cannot be located, RSTS/E generates the error Can't find file or account (ERR=5). That same error condition also applies to other, generically similar access operations. Thus, if a program tries to send a message to another program and the system cannot find the proper entry 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, cannot be trapped by a program. Table C-3 lists these errors.

Table C-3: Nontrappable 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 that your program cannot control and that should not occur on a normal system. For example, the error ??Disk error during swap indicates a hardware problem. The system does not return control to the program. The error condition itself, however, can be either transient or recurring.

Bring these errors to the attention of your system manager for further investigation. These errors are recoverable in the strict sense that the monitor can take corrective action. However, the BASIC-PLUS run-time system does not return control to your program.

User Recoverable Errors

Table C-4 lists the user recoverable errors. The notations (C) and (SPR) follow some error explanations. See Table C-2 for an explanation of these special abbreviations.

Table C-4: User Recoverable Errors

Message and Meaning	ERR Value
(SYSTEM INSTALLATION NAME) The error code 0 is associated with the system installation name. System programs use this to print identification lines.	0
??BAD DIRECTORY FOR DEVICE o The directory of the device referenced is in an unreadable format. o 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 MOUNT command to set the correct format default or change the format specification in the MODE option of the OPEN statement.	1
?ILLEGAL FILE NAME o The specified file name or type is not acceptable. It contains unacceptable characters or violates the file specification format. o The CCL command to be added begins with a number or contains an illegal character.	2

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
?ACCOUNT OR DEVICE IN USE o An attempt to reassign or dismount the device fails because the device is open or has one or more open files. o The account to be deleted has one or more files and must be zeroed before being deleted. o The run-time system to be deleted is currently loaded in memory and in use. o Output to a pseudo keyboard cannot be done unless the device is in KB wait state. o An echo control field cannot be declared while another field is currently active and the system has input characters for your program. o The CCL command to be added already exists. o The disk being accessed was mounted /NOSHARE by another user.	3
?NO ROOM FOR USER ON DEVICE o You have already used the available storage space. o The device as a whole is too full to accept further data. o The directory is full. o You are sending a message to a message receiver that already has its maximum number of messages pending.	4
?CAN'T FIND FILE OR ACCOUNT o The specified file or account number was not found on the specified device. o The CCL command to be deleted does not exist.	5
?NOT A VALID DEVICE The device specification supplied is not valid for one of the following reasons: o The unit number or its type is not configured on the system. o The specification is logical and untranslatable because a physical device is not associated with it.	6

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
?I/O CHANNEL ALREADY OPEN You tried to open an I/O channel that the program had already opened. Note that this error cannot occur in BASIC-PLUS or BASIC-PLUS-2, since both automatically close and then reopen the channel. (SPR)	7
?DEVICE NOT AVAILABLE The specified device exists on the system but you cannot assign or open it for one of the following reasons: o The device is currently reserved by another job. o The device requires privileges for ownership that you do not have. o The system manager has disabled the device or its controller. o The device is a keyboard line for pseudo keyboard use only.	8
?I/O CHANNEL NOT OPEN You tried to perform I/O on one of the twelve channels that the program has not previously opened.	9
?PROTECTION VIOLATION You cannot perform the requested operation because the operation is illegal (such as input from a line printer) or because you do not have the necessary privileges (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-PLUS source file without an END statement is called into memory.	11
??FATAL SYSTEM I/O FAILURE An I/O error has occurred at the system level. You have 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 this appendix.	12

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
?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 Check hardware condition of the requested device. Possible causes of this error include a line printer out of paper or device being offline.	14
?KEYBOARD WAIT EXHAUSTED Time that the WAIT statement requests has been exhausted with no input received from the specified keyboard.	15
?NAME OR ACCOUNT NOW EXISTS Either you tried to rename a file with the name of a file that already exists, or the system manager tried to create an account number that is already in 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
?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
?DEVICE IS RESTRICTED The specified disk pack is marked restricted by another user. You need DEVICE privilege to access it.	22

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
?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
?ACCOUNT DOES NOT EXIST You tried to create a file in a nonexistent account on a private disk.	24
%DISK PACK NEEDS REBUILDING This is a nonfatal disk mounting error. Use the DCL MOUNT command.	25
??DISK PACK MOUNT ERROR This is a fatal disk mounting error. The disk cannot be successfully mounted. The disk structure is corrupt or it is not a RSTS disk. Use DSKINT to put the RSTS structure on the disk.	26
?I/O TO DETACHED KEYBOARD This error has the following possible causes: o The job is detached and one of the simple terminal SYS system function calls (function codes 0,1,2,3,4 or 11) is attempted for the job's console terminal (KB:). o The job is detached and an open is attempted using the device name "KB:". o Any I/O operation, such as INPUT, PRINT, GET, or PUT, is attempted to a terminal on a hung-up dial-up line and the terminal is neither the job's console terminal nor the terminal from which the job is detached. Opening a dial-up line that is currently hung up does not cause an error.	27

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
o The job is detached and I/O is attempted to a terminal that was opened with MODE 16%. (When MODE 16% is not specified, the job hibernates when it becomes detached and terminal I/O is attempted.) Note that the system places a detached job in hibernation when an I/O request is issued to the job's console (KB:) terminal or to any channel on which that terminal is open. Thus, hibernation can occur with local terminals if the job detaches. Hibernation can also occur on dial-up lines if the job detaches or if the line is hung up, causing the system to automatically detach the job.	
PROGRAMMABLE ^C TRAP A CTRL/C was typed while an ON ERROR GOTO statement was in effect and programmable CTRL/C trapping was enabled.	28
??CORRUPTED FILE STRUCTURE	29
?DEVICE NOT FILE STRUCTURED An attempt is made to access a device other than a disk, DECtape, or magnetic tape as a file-structured device. This error occurs, for example, when you attempt to get a directory listing of a nondirectory device.	30
?ILLEGAL BYTE COUNT FOR I/O o The buffer size specified in the RECORDSIZE option of the OPEN statement or the COUNT option of the PUT statement is not a multiple of the block size of the device you are using for I/O or is illegal for the device. o You tried to run a compiled file that has improper size due to incorrect transfer procedure. o You specified illegal parameters.	31
?NO BUFFER SPACE AVAILABLE o You accessed a file and the monitor requires one small buffer to complete the request but there is no buffer available. o The program is sending a message and a small buffer is not available for the operation.	32

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
??ODD ADDRESS TRAP This error occurs when you attempt to reference a nonexistent address, reference an odd address using a word instruction, or perform a PEEK function with an odd or nonexistent parameter. If you get this error for any other reason, report it to your system manager.	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 the discussion at beginning of this appendix.	34
??MEMORY MANAGEMENT TRAP You specified an illegal monitor address in the PEEK function. If you get this error for any other reason, report it to your system manager.	35
??SP STACK OVERFLOW An attempt to extend the hardware stack beyond its legal size is encountered. See the discussion at beginning of this appendix. (SPR)	36
??DISK ERROR DURING SWAP A hardware error occurs when your job is swapped into or out of memory. The contents of your job area are lost, but the job remains logged in to the system and is reinitialized to run the NONAME program. Report such occurrences to the system manager. See the discussion at beginning of this appendix.	37
??MEMORY PARITY FAILURE A parity error was detected in the memory occupied by this job. See the discussion at beginning of this appendix.	38
?MAGTAPE SELECT ERROR When access to a magnetic tape drive was attempted, the selected unit was found to be off line.	39

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
?MAGTAPE RECORD LENGTH ERROR When performing input from magnetic tape, the record on tape was longer than the buffer designated to handle the record.	40
??NON-RES RUN-TIME SYSTEM A hardware error occurred when loading a run-time system or resident library for your job. Report such occurrences to the system manager.	41
?VIRTUAL BUFFER TOO LARGE Virtual array buffers must be 512 bytes long.	42
?VIRTUAL ARRAY NOT ON DISK A nondisk device is open on the channel on which the virtual array is referenced.	43
?MATRIX OR ARRAY TOO BIG Memory array size is too large.	44
?VIRTUAL ARRAY NOT YET OPEN You tried to use a virtual array before opening the corresponding disk file.	45
?ILLEGAL I/O CHANNEL You tried to open a file on an I/O channel outside the range of the integer numbers 1 to 12.	46
?LINE TOO LONG You tried to input a line longer than 255 characters (which includes any line terminator). The buffer overflows.	47
%FLOATING POINT ERROR You tried to use a computed floating-point number outside the range 1E-38<n<1E38. 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

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
%DATA FORMAT ERROR A READ or INPUT statement detected data in an illegal format. For example, 1..2 is an improperly formed number, 1.3 is an improperly formed integer, and "HELLO" "THERE" is an illegal string. (C)	50
%INTEGER ERROR You tried to use a computed integer outside the range -32768<n<32767. For example, you tried 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 causes this error. Value returned is the argument as passed to the function. (C)	53
%IMAGINARY SQUARE ROOTS You tried to take the square root of a number less than zero. The value returned is the square root of the absolute value of the argument. (C)	54
?SUBSCRIPT OUT OF RANGE You tried to reference an array element beyond the number of elements created for the array when it was dimensioned.	55
?CAN'T INVERT MATRIX You tried to invert a singular or nearly singular matrix.	56
?OUT OF DATA The DATA list was exhausted and a READ requested additional data.	57

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
?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 32767 or below -32768.	60
%DIVISION BY 0 Your program attempted to divide some quantity by zero. If no transfer is made to an error handling routine, the result is zero. (C)	61
?NO RUN-TIME SYSTEM The run-time system referenced has not been added to the system list of run-time systems.	62
?FIELD OVERFLOWS BUFFER You tried to use FIELD to allocate more space than exists in the specified buffer.	63
?NOT A RANDOM ACCESS DEVICE You tried to perform random access I/O to a nonrandom access device.	64
?ILLEGAL MAGTAPE() USAGE Improper use of the MAGTAPE function.	65
?MISSING SPECIAL FEATURE o Your program uses a BASIC-PLUS feature not present on the given installation. o You attempted to use MODE 512% on a line printer that has 8 bit capabilities set. o You attempted to use a DECnet function but DECnet/E is not installed.	66

Table C-4: User Recoverable Errors (Cont.)

Message and Meaning	ERR Value
?ILLEGAL SWITCH USAGE o 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.) o A file specification switch is not the last element in a file specification or is missing a colon or an argument.	67
?END OF VOLUME You are reading an ANSI magnetic tape and reached an end-of-volume (EOV) label. The message indicates that the data continues on another volume. (See the section "Processing Multivolume ANSI Magnetic Tape Files," in Chapter 2.)	68
?QUOTA EXCEEDED You exceeded the logged-in disk quota for your account. Or, you exceeded some other quota with a SYS function call (job, detached job, RIB, or message quota.	69

Nonrecoverable Errors

Table C-5 lists the nonrecoverable errors. The notations (C) and (SPR) follow some error explanations. See Table C-2 for an explanation of these special abbreviations.

Table C-5: Nonrecoverable Errors

Message and Meaning
?ARGUMENTS DON'T MATCH Arguments in a function call do not match, in number or in type, the arguments defined for the function.
?BAD LINE NUMBER PAIR Line numbers specified in a LIST or DELETE command were formatted incorrectly.

Table C-5: Nonrecoverable Errors (Cont.)

Message and Meaning
?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 with CONT or CCONT.
?DATA TYPE ERROR Incorrect use of floating-point, integer, or character string variable or constant where some other data type was necessary.
?DEF WITHOUT FNEND A second DEF statement was encountered in the processing of a user function without an FNEND.
?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 offline, 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 file.
?EXPRESSION TOO COMPLICATED This error usually occurs when parentheses have been nested too deeply. The depth allowed depends on the individual expression.
?FILE EXISTS-RENAME/REPLACE A file of the name specified in a SAVE command already exists. To save the current program with the name specified, use REPLACE or RENAME followed by SAVE.
?FNEND WITHOUT DEF An FNEND statement was encountered in your program before a DEF statement was seen.
?FNEND WITHOUT FUNCTION CALL A FNEND statement was encountered in your program before a function call was executed.

Table C-5: Nonrecoverable Errors (Cont.)

Message and Meaning
?FOR WITHOUT NEXT A FOR statement was encountered in your program without a corresponding NEXT statement to terminate the loop.
?ILLEGAL CONDITIONAL CLAUSE You used an incorrectly formatted conditional expression.
?ILLEGAL DEF NESTING The range of one function definition crosses the range of another function definition.
?ILLEGAL DUMMY VARIABLE One of the variables in the dummy variable list of a user-defined function is not a legal variable name.
?ILLEGAL EXPRESSION Double operators, missing operators, mismatched parentheses, or some similar error was 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 of incorrect format.
?ILLEGAL IF STATEMENT You used an incorrectly formatted IF statement.
?ILLEGAL IN IMMEDIATE MODE You entered a statement in immediate mode that can only be executed as part of a program.
?ILLEGAL LINE NUMBER(S) A line number reference is outside the range 1<n<32767.
?ILLEGAL MODE MIXING String and numeric operations cannot be mixed.
?ILLEGAL STATEMENT An attempt was made to execute a statement that did not compile without errors.

Table C-5: Nonrecoverable Errors (Cont.)

Message and Meaning
?ILLEGAL SYMBOL An unrecognizable character was encountered. For example, a line consisting of a % character causes this error.
?ILLEGAL VERB The verb portion of the BASIC-PLUS statement cannot be recognized.
%INCONSISTENT FUNCTION USAGE A function is defined with a certain number of arguments but is referenced elsewhere with a different number of arguments. Correct 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.
?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 o During an OLD operation, the job's private memory size maximum was reached. o 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 o An attempt is made to use one of the statement modifiers (FOR, WHILE, UNTIL, IF, or UNLESS) incorrectly. o An OPEN statement modifier, such as a RECORDSIZE, CLUSTERSIZE, FILESIZE, or MODE option, is not in the correct order.

Table C-5: Nonrecoverable Errors (Cont.)

Message and Meaning
?NEXT WITHOUT FOR A NEXT statement was encountered in your program without a previous FOR statement.
?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 was made to load a nonprivileged compiled program that 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. A user who was not logged into the system typed something other than a legal, logged-out command.
?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.

Table C-5: Nonrecoverable Errors (Cont.)

Message and Meaning
??PROGRAM LOST-SORRY A fatal system error has occurred that caused your program to be lost. This error can indicate hardware problems or use of an improperly compiled program. See the next section for more information.
?REDIMENSIONED ARRAY Use of an array or matrix within your 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 with the ON ERROR GOTO statement.
?RETURN WITHOUT GOSUB RETURN statement is encountered in your program when a previous GOSUB statement was not executed.
%SCALE FACTOR INTERLOCK o You set a new scale factor and then executed a program that was compiled (that is, translated) using a different scale factor. The program runs, but BASIC-PLUS uses the scale factor in effect when the program was compiled. To cause BASIC-PLUS to compile the program with the new scale factor, use REPLACE and OLD. o You set a new scale factor and then entered an immediate mode statement. Immediate mode statements are always compiled using the current scale factor. The new scale factor will take effect when you use the NEW or OLD command or run a program from its source file. See the *BASIC-PLUS Language Manual* for more information. (C)
?STATEMENT NOT FOUND Reference is made in the program to a line number that is not in the program.
STOP STOP statement was executed. You 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.

Table C-5: Nonrecoverable Errors (Cont.)

Message and Meaning
?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 can have up to five arguments.
?UNDEFINED FUNCTION CALLED BASIC-PLUS interpreted some statement component as a function call for which there is no defined function (system or user).
?WHAT? You entered a command or immediate mode statement that BASIC-PLUS cannot process. An illegal verb or improper format error is most likely.
?WRONG MATH PACKAGE Program was compiled on a system with either the two-word or four-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.

BASIC-PLUS-2 Errors

Table C-6 lists the BASIC-PLUS-2 errors. For explanations of these error messages, see the appropriate BASIC-PLUS-2 documentation.

Table C-6: BASIC-PLUS-2 Errors

Message	
?1st arg to SEQ$ > 2nd	?Key larger than record
?Arrays must be same dim	?Key not changeable
?Arrays must be square	?Key size too large
?Argument out of bounds	?Move overflows buffer

Table C-6: BASIC-PLUS-2 Errors (Cont.)

Message	
?Bad record identifier	?Negative fill or string len
?Bad RECORDSIZE on OPEN	?Negative TAB not allowed
?Cannot change array dims	?Network operation rejected
?Cannot open file	?No current record
?Cannot position to EOF	?No fields in image
?CHAIN to non-existent line	?No file name
%Decimal overflow	?No primary key specified
?Directive error	?No support for op in task
?Duplicate key detected	?Node name error
?Error trap needs RESUME	?Not at end of file
?Exponentiation error	?Null image
?FILE ACP failure	?Numeric image for string
?File attributes not matched	?OPEN Error - file corrupted
?File is locked	?Primary key out of sequence
?Floating overflow	?Record already exists
?Floating underflow	?Record/bucket locked
?Index not initialized	?Record has been deleted
?Invalid file options	?Record LOCK failed
?Illegal key attributes	?Record not found
?Invalid key of reference	?RECORD number exceeds max
?Illegal ALLOW clause	?Record on file too big
?Illegal exit from DEF*	%RECORDSIZE overflows MAP
?Illegal operation	?RECORDTYPES not matched

Table C-6: BASIC-PLUS-2 Errors (Cont.)

Message	
?Illegal or illogical access	?Recursive subroutine call
?Illegal record format	?REMAP overflows buffer
?Illegal record lock clause	?RRV not fully updated
?Illegal record on file	?Size of record invalid
?Illegal RESUME to SUBR	?String image for numeric
?Illegal string image	?String too long
?Illegal subroutine return	?Tape BOT detected
?Illegal usage	?Tape not ANSI labeled
?Illegal usage for device	?Tape records not ANSI
?Illogical record accessing	?Terminal fmt file required
?Improper error handling	?TIME limit exceeded
?Indexed not fully optimized	?Too much data in record
?Invalid RFA field	?Unaligned REMAP variable
?Key field beyond record end	?Unexpired file date

The ??Program Lost-Sorry Error

The ??Program lost-sorry error occurs when BASIC-PLUS tries to run a program and cannot. BASIC-PLUS clears the job image from memory and returns control to the user. If possible, BASIC-PLUS prints a second message that provides more information about what caused the program to be lost. In several cases, however, only the ??Program lost-sorry message is printed, and the system manager must check the error log to determine the cause. Always report a ??Program lost-sorry message and its associated message (if printed) to your system manager.

The ??Program lost-sorry error has four possible causes:

- A checksum error occurs on a .BAC file. (A checksum error is usually the result of a hardware problem.)
- An unrecoverable disk error occurs while BASIC-PLUS is reading a .BAC file.
- BASIC-PLUS tries to load a .BAC file of incorrect size.
- BASIC-PLUS tries to run a file whose stored version number does not match the current BASIC-PLUS run-time system's version number.

You can often recover by recompiling the program from its source file and running it again. To recompile the program:

1. Use the OLD command to translate (compile) the program from its source file. OLD places the translated program in memory.
2. Use the COMPILE command to create a new .BAC file that contains the translated image.

The next four sections describe each of the possible causes in more detail.

Checksum Error on a .BAC File

A "checksum" is a numeric quantity that is used to detect errors. When you save a translated program in a disk file with the COMPILE command, BASIC-PLUS computes a checksum and stores it in the file. BASIC-PLUS computes another checksum when it loads the .BAC file from disk. An error occurs if the computed and stored checksums do not match.

If the checksums are not equal, BASIC-PLUS produces an error to be logged by the RSTS/E monitor, returns the ??Program lost-sorry error to the user, and aborts program execution.

Checksum errors are usually caused by a disk error. The disk error may have occurred when you created the .BAC file or it may have occurred while BASIC-PLUS was reading the .BAC file into memory.

You can recover by recompiling the program and running it again.

Unrecoverable Disk Error Reading a .BAC File

The ??Program lost-sorry error also results when an unrecoverable disk error occurs while BASIC-PLUS is loading a .BAC file into memory. Unrecoverable disk errors can result from bad disk blocks, dust, problems with the disk drive, or a transient hardware problem in the disk subsystem. Sometimes these errors produce an error such as ??Disk error during swap, which is logged in the system error logging file.

Recompiling the program may correct the problem. Be sure to report the problem to your system manager.

Incorrect .BAC File Size

A .BAC file must be between 2K and 16K words (inclusive). In addition, the number of blocks in the file must be an integer that is one less than a multiple of 4.

If the size of the .BAC file does not follow these rules, BASIC-PLUS prints two messages when it tries to load the file into memory: ??Program lost-sorry and ?Illegal byte count for I/O. These errors are not logged in the system error logging file.

To correct the problem, recompile the program.

Unmatched Version Numbers

When you use COMPILE to save a translated program, BASIC-PLUS writes the current version number of the BASIC-PLUS run-time system into the .BAC file. When BASIC-PLUS runs or chains to a .BAC file, it checks the version number stored in the file against the version number of the run-time system being used. If the version numbers do not match, the ??Program lost-sorry error results.

Consult the *RSTS/E Release Notes* to find out whether recompilation of current programs is necessary for a new version of BASIC-PLUS.

Software Performance Report Guidelines

The Software Performance Report (SPR) forms let you report problems with DIGITAL software. Before submitting an SPR, make sure 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, describe the problem as completely as possible. The following list contains the minimal information to include in 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 the list of privileges assigned to the account.
- 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 you submit a crash dump, it must include the output of the crash dump analysis program. Machine-readable submissions must include the CRASH.SYS file and monitor .SIL file in use at the time of the crash. For problems related to run-time systems, include the .RTS file.

Appendix D

Radix-50 and ASCII Character Sets

Radix-50 Character Set

Many items in RSTS/E, such as file names and file types, are stored in Radix-50 format. This format allows three characters to be stored as a two-byte integer (one 16-bit word). The RAD$ function converts a Radix-50 word to its three-character representation. In addition, the file name string scan SYS calls convert three-character strings to Radix-50 format.

The following chart shows the complete set of characters that can be represented in Radix-50 format, their ASCII octal equivalents, and the Radix-50 value for each character:

Character	ASCII Octal Equivalent	Radix-50 Equivalent (octal)
space	40	0
A-Z	101-132	1-32
$	44	33
.	56	34
?	77	35
0-9	60-71	36-47

The value of a character in its two-byte Radix-50 representation depends on its position in the string. 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 three-character string, add the values of the three characters. For example, the maximum Radix-50 value (representing the character string 999) is:

```
47*50^2 + 47*50^1 + 47*50^0 = 174777
```

Table D-1 provides an easy way to translate between the ASCII character set and its Radix-50 equivalents based on position within a string.

For convenience, the table contains the decimal value of each character as well as its octal value. (You can find the decimal value of a Radix-50 word using the same technique shown here for octal.)

Table D-1: Radix-50 Character Positions

Single or 1st Char.	Octal	Decimal	2nd Char.	Octal	Decimal	3rd Char.	Octal	Dec.
A	003100	1600	A	000050	40	A	000001	1
B	006200	3200	B	000120	80	B	000002	2
C	011300	4800	C	000170	120	C	000003	3
D	014400	6400	D	000240	160	D	000004	4
E	017500	8000	E	000310	200	E	000005	5
F	022600	9600	F	000360	240	F	000006	6
G	025700	11200	G	000430	280	G	000007	7
H	031000	12800	H	000500	320	H	000010	8
I	034100	14400	I	000550	360	I	000011	9
J	037200	16000	J	000620	400	J	000012	10
K	042300	17600	K	000670	440	K	000013	11
L	045400	19200	L	000740	480	L	000014	12
M	050500	20800	M	001010	520	M	000015	13
N	053600	22400	N	001060	560	N	000016	14
O	056700	24000	O	001130	600	O	000017	15
P	062000	25600	P	001200	640	P	000020	16
Q	065100	27200	Q	001250	680	Q	000021	17
R	070200	28800	R	001320	720	R	000022	18
S	073300	30400	S	001370	760	S	000023	19
T	076400	32000	T	001440	800	T	000024	20
U	101500	33600	U	001510	840	U	000025	21
V	104600	35200	V	001560	880	V	000026	22
W	107700	36800	W	001630	920	W	000027	23
X	113000	38400	X	001700	960	X	000030	24
Y	116100	40000	Y	001750	1000	Y	000031	25
Z	121200	41600	Z	002020	1040	Z	000032	26
$	124300	43200	$	002070	1080	$	000033	27
.	127400	44800	.	002140	1120	.	000034	28
?	132500	46400	?	002210	1160	?	000035	29
0	135600	48000	0	002260	1200	0	000036	30
1	140700	49600	1	002330	1240	1	000037	31
2	144000	51200	2	002400	1280	2	000040	32
3	147100	52800	3	002450	1320	3	000041	33
4	152200	54400	4	002520	1360	4	000042	34
5	155300	56000	5	002570	1400	5	000043	35

Table D-1: Radix-50 Character Positions (Cont.)

Single or 1st Char.	Octal	Decimal	2nd Char.	Octal	Decimal	3rd Char.	Octal	Dec.
6	160400	57600	6	002640	1440	6	000044	36
7	163500	59200	7	002710	1480	7	000045	37
8	166600	60800	8	002760	1520	8	000046	38
9	171700	62400	9	003030	1560	9	000047	39

A three-character string is stored left to right in the Radix-50 word. For example, given the ASCII string X2B, you can compute the Radix-50 representation as follows:

```
X   = 113000(octal)
2   = 002400(octal)
B   = 000002(octal)

X2B = 115402(octal)
```

(Note that addition is done in octal.)

To represent a three-character string in Radix-50 format:

- Place the first character of a string (or a single character) in the leftmost position of the Radix-50 word. Thus, for the character X, multiply its representation 30(octal) by 50^2 to give 113000(octal), the value shown in Table D-1 for X when it is the first character.

- Place the second character in a string in the next position to the right. For the character 2 (in the second position), multiply its representation 40(octal) by 50^1 to give 002400, the value shown in Table D-1 for 2 when it is the second character.

- Place the third character in the rightmost position. For the character B (in the third position), multiply its representation by 50^0 (which is 1) to give 000002, the value shown in Table D-1 for B when it is the third character.

To get the full octal value of the Radix-50 word, add the value of each character by its position in the string.

ASCII Character Codes

Table D-2 lists the ASCII characters and their decimal and octal values.

Table D-2: ASCII Character Codes

ASCII			
Decimal	Octal	Character	Remarks
0	000	NUL	Null, FILL character
1	001	SOH	CTRL/A
2	002	STX	CTRL/B
3	003	ETX	CTRL/C
4	004	EOT	End of transmission, CTRL/D
5	005	ENQ	CTRL/E
6	006	ACK	CTRL/F
7	007	BEL	Bell, CTRL/G
8	010	BS	Backspace, CTRL/H
9	011	HT	Horizontal tab, CTRL/I
10	012	LF	Line feed, CTRL/J
11	013	VT	Vertical tab, CTRL/K
12	014	FF	Form feed, page, CTRL/L
13	015	CR	Carriage return, CTRL/M
14	016	SO	CTRL/N
15	017	SI	CTRL/O
16	020	DLE	CTRL/P
17	021	DC1	CTRL/Q*, XON
18	022	DC2	CTRL/R
19	023	DC3	CTRL/S**, XOFF
20	024	DC4	CTRL/T
21	025	NAK	CTRL/U
22	026	SYN	CTRL/V
23	027	ETB	CTRL/W
24	030	CAN	CTRL/X
25	031	EM	CTRL/Y
26	032	SUB	CTRL/Z, end of file
27	033	ESC	Escape***
28	034	FS	File Separator
29	035	GS	Group Separator
30	036	RS	Record Separator
31	037	US	Unit Separator
32	040	SP	Space or blank
33	041	!	Exclamation point

Table D-2: ASCII Character Codes (Cont.)

Decimal	Octal	Character	Remarks
34	042	"	Quotation mark
35	043	#	Number sign
36	044	$	Dollar sign
37	045	%	Percent sign
38	046	&	Ampersand
39	047	'	Apostrophe
40	050	(	Left parenthesis
41	051	)	Right parenthesis
42	052	*	Asterisk
43	053	+	Plus
44	054	,	Comma
45	055	-	Hyphen or minus
46	056	.	Period or decimal point
47	057	/	Slash
48	060	0	Zero
49	061	1	One
50	062	2	Two
51	063	3	Three
52	064	4	Four
53	065	5	Five
54	066	6	Six
55	067	7	Seven
56	070	8	Eight
57	071	9	Nine
58	072	:	Colon
59	073	;	Semicolon
60	074	<	Left angle bracket, "less than" sign
61	075	=	Equal sign
62	076	>	Right angle bracket, "greater than" sign
63	077	?	Question mark
64	100	@	At sign
65	101	A	Uppercase A
66	102	B	Uppercase B
67	103	C	Uppercase C
68	104	D	Uppercase D
69	105	E	Uppercase E
70	106	F	Uppercase F
71	107	G	Uppercase G
72	110	H	Uppercase H
73	111	I	Uppercase I
74	112	J	Uppercase J
75	113	K	Uppercase K
76	114	L	Uppercase L

Table D-2: ASCII Character Codes (Cont.)

Decimal	Octal	Character	Remarks
77	115	M	Uppercase M
78	116	N	Uppercase N
79	117	O	Uppercase O
80	120	P	Uppercase P
81	121	Q	Uppercase Q
82	122	R	Uppercase R
83	123	S	Uppercase S
84	124	T	Uppercase T
85	125	U	Uppercase U
86	126	V	Uppercase V
87	127	W	Uppercase W
88	130	X	Uppercase X
89	131	Y	Uppercase Y
90	132	Z	Uppercase Z
91	133	[	Left square bracket
92	134	\	Backslash
93	135	]	Right square bracket
94	136	^	Circumflex
95	137	_	Underscore
96	140	`	Grave accent
97	141	a	Lowercase a
98	142	b	Lowercase b
99	143	c	Lowercase c
100	144	d	Lowercase d
101	145	e	Lowercase e
102	146	f	Lowercase f
103	147	g	Lowercase g
104	150	h	Lowercase h
105	151	i	Lowercase i
106	152	j	Lowercase j
107	153	k	Lowercase k
108	154	l	Lowercase l
109	155	m	Lowercase m
110	156	n	Lowercase n
111	157	o	Lowercase o
112	160	p	Lowercase p
113	161	q	Lowercase q
114	162	r	Lowercase r
115	163	s	Lowercase s
116	164	t	Lowercase t
117	165	u	Lowercase u
118	166	v	Lowercase v
119	167	w	Lowercase w
120	170	x	Lowercase x
121	171	y	Lowercase y

Table D-2: ASCII Character Codes (Cont.)

Decimal	Octal	Character	Remarks
122	172	z	Lowercase z
123	173	{	Left brace
124	174	|	Vertical line
125	175	}	Right brace ***
126	176	~	Tilde ***
127	177	DEL	Delete
128	200		Reserved
129	201		Reserved
130	202		Reserved
131	203		Reserved
132	204	IND	Index
133	205	NEL	New line
134	206	SSA	
135	207	ESA	
136	210	HTS	Horizontal tab set
137	211	HTJ	
138	212	VTS	Vertical tab set
139	213	PLD	Partial line down
140	214	PLU	Partial line up
141	215	RI	Reverse Index
142	216	SS2	Single shift 2
143	217	SS3	Single shift 3
144	220	DCS	Device control string
145	221	PU1	
146	222	PU2	
147	223	STS	
148	224	CCH	
149	225	MW	
150	226	SPA	
151	227	EPA	
152	230		Reserved
153	231		Reserved
154	232		Reserved
155	233	CSI	Control sequence introducer
156	234	ST	String terminator
157	235	OSC	
158	236	PM	
159	237	APC	
160	240		Reserved
161	241	¡	Inverted exclamation point
162	242	¢	Cent sign
163	243	£	Pound sign
164	244		Reserved
165	245	¥	Yen sign
166	246		Reserved

Table D-2: ASCII Character Codes (Cont.)

Decimal	Octal	Character	Remarks
167	247	§	Section sign
168	250	¤	General currency sign
169	251	©	Copyright sign
170	252	ª	Feminine ordinal indicator
171	253	«	Angle quotation mark left
172	254		Reserved
173	255		Reserved
174	256		Reserved
175	257		Reserved
176	260	°	Degree sign
177	261	±	Plus/minus sign
178	262	²	Superscript 2
179	263	³	Superscript 3
180	264		Reserved
181	265	µ	Micro sign
182	266	¶	Paragraph sign, pilcrow
183	267	·	Middle dot
184	270		Reserved
185	271	¹	Superscript 1
186	272	º	Masculine ordinal indicator
187	273	»	Angle quotation mark right
188	274	¼	Fraction one quarter
189	275	½	Fraction one half
190	276		Reserved
191	277	¿	Inverted question mark
192	300	À	Uppercase A with grave accent
193	301	Á	Uppercase A with acute accent
194	302	Â	Uppercase A with circumflex accent
195	303	Ã	Uppercase A with tilde
196	304	Ä	Uppercase A with diaeresis or umlaut mark
197	305	Å	Uppercase A with ring
198	306	Æ	Uppercase A with dipthong
199	307	Ç	Uppercase C with cedilla
200	310	È	Uppercase E with grave accent
201	311	É	Uppercase E with acute accent
202	312	Ê	Uppercase E with circumflex accent
203	313	Ë	Uppercase E with diaeresis or umlaut mark
204	314	Ì	Uppercase I with grave accent
205	315	Í	Uppercase I with acute accent
206	316	Î	Uppercase I with circumflex accent
207	317	Ï	Uppercase I with diaeresis or umlaut mark
208	320		Reserved

Table D-2: ASCII Character Codes (Cont.)

Decimal	Octal	Character	Remarks
209	321	Ñ	Uppercase N with tilde
210	322	Ò	Uppercase O with grave accent
211	323	Ó	Uppercase O with acute accent
212	324	Ô	Uppercase O with circumflex accent
213	325	Õ	Uppercase O with tilde
214	326	Ö	Uppercase O with diaeresis or umlaut mark
215	327	Œ	Uppercase OE ligature
216	330	Ø	Uppercase O with slash
217	331	Ù	Uppercase U with grave accent
218	332	Ú	Uppercase U with acute accent
219	333	Û	Uppercase U with circumflex accent
220	334	Ü	Uppercase U with diaeresis or umlaut mark
221	335	Ÿ	Uppercase Y with diaeresis or umlaut mark
222	336		Reserved
223	337	ß	German lowercase sharp s
224	340	à	Lowercase a with grave accent
225	341	á	Lowercase a with acute accent
226	342	â	Lowercase a with circumflex accent
227	343	ã	Lowercase a with tilde
228	344	ä	Lowercase a with diaeresis or umlaut mark
229	345	å	Lowercase a with ring
230	346	æ	Lowercase ae dipthong
231	347	ç	Lowercase c with cedilla
232	350	è	Lowercase e with grave accent
233	351	é	Lowercase e with acute accent
234	352	ê	Lowercase e with circumflex accent
235	353	ë	Lowercase e with diaeresis or umlaut mark
236	354	ì	Lowercase i with grave accent
237	355	í	Lowercase i with acute accent
238	356	î	Lowercase i with circumflex accent
239	357	ï	Lowercase i with diaeresis or umlaut mark
240	360		Reserved
241	361	ñ	Lowercase n with tilde
242	362	ò	Lowercase o with grave accent
243	363	ó	Lowercase o with acute accent
244	364	ô	Lowercase o with circumflex accent
245	365	õ	Lowercase o with tilde

Table D-2: ASCII Character Codes (Cont.)

Decimal	Octal	Character	Remarks
246	366	ö	Lowercase o with diaeresis or umlaut mark
247	367	œ	Lowercase oe ligature
248	370	ø	Lowercase o with slash
249	371	ù	Lowercase u with grave accent
250	372	ú	Lowercase u with acute accent
251	373	û	Lowercase u with circumflex accent
252	374	ü	Lowercase u with diaeresis or umlaut mark
253	375	ÿ	Lowercase y with diaeresis or umlaut mark
254	376		Reserved
255	377		Reserved

* CTRL/Q, or XON, resumes output if the TTSYNC terminal characteristic is set.

** CTRL/S, or XOFF, stops output if the TTSYNC terminal characteristic is set.

*** ALTMODE(ASCII 125) or PREFIX (ASCII 126) keys, which appear on some terminals, are translated internally into ESCAPE if the ALT MODE terminal characteristic is set.

Appendix E

Device Handler Index

Table E-1 lists the handler indexes for each device type used on RSTS/E. The handler index is an internal index into system device tables that the system uses to identify device families. For example, the handler index is used in SPEC% functions to ensure that the system operates on the correct device.

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	Pseudo Keyboards
18	Flexible Diskettes [RX01, RX02]
20	RJ2780
22	Null Device

Table E-1: Handler Index (Cont.)

Index	Device
24	DMC11/DMR11 DDCMP Interface
32	KMC11
34	IBM Interface
38	DMP11/DMV11

Appendix F

Monitor Directives

The *RSTS/E System Directives Manual* describes monitor directives for MACRO programmers. Many of these directives correspond to SYS calls described in Chapter 7 of this manual.

Table F-1 lists the SYS call to FIP codes and the corresponding monitor directives. For information on the use of these directives, see the *RSTS/E System Directives Manual*.

Table F-1: Monitor Directives

MACRO Mnemonic	Function Code(FO)	Function Name
UU.TB3	-29	Get monitor tables - part III.
UU.SPL	-28	Spooling.
UU.DMP	-27	Snap shot dump.
UU.FIL	-26	File utility functions.
UU.ATR	-25	Read/Write file attributes; Read pack attributes; Read/Write/Delete account attributes
UU.CCL	-24	Add/delete CCL command.
(.FSS)	-23	Terminating file name string scan.
(.SET)	-22	Set special run priority.
(.SET/.CLEAR)	-21	Drop/regain (temporary) privileges.
(.SET/.CLEAR)	-20	Lock/unlock job in memory.

Table F-1: Monitor Directives (Cont.)

MACRO Mnemonic	Function Code(FO)	Function Name
UU.LOG	-19	Set number of logins.
UU.RTS	-18	Add/Remove/Unload run-time system; Add/Remove/Unload resident library; Create dynamic region.
UU.NAM	-17	Name run-time system.
UU.DIE	-16	Shut down system.
UU.ACT	-15	Accounting dump.
UU.DAT	-14	Change system date/time.
UU.PRI	-13	Change priority/run burst/job size.
UU.TB2	-12	Get monitor tables - part II.
UU.BCK	-11	Change file backup statistics.
(.FSS)	-10	File name string scan.
UU.HNG	-9	Hang up a dataset.
UU.FCB	-8	Get open channel statistics.
-	-7	Enable CTRL/C trap.
UU.POK	-6	Poke memory.
(.SPEC)	-5	Broadcast to terminal.
(.SPEC)	-4	Force input to terminal.
UU.TB1	-3	Get monitor tables - part I.
UU.NLG	-2	Disable logins.
UU.YLG	-1	Enable logins.
UU.PAS	0	Create user account.
UU.DLU	1	Delete user account.
-	2	Reserved.

Table F-1: Monitor Directives (Cont.)

MACRO Mnemonic	Function Code(FO)	Function Name
UU.MNT	3	Disk pack status.
UU.LIN	4	Login; Verify password.
UU.BYE	5	Logout.
UU.ATT	6	Attach; Reattach; Swap console.
UU.DET	7	Detach.
UU.CHU	8	Change quota/expiration date/password;Set password; Kill job; Disable terminal.
UU.ERR	9	Return error messages.
UU.ASS (.UUO) UU.ASS (.ULOG)	10	Allocate/Reallocate device; Assign user logical.
UU.DEA (.UUO) UU.DEA (.ULOG)	11	Deallocate device; Deassign user logical.
UU.DAL (.UUO) UU.DAL (.ULOG)	12	Deallocate all devices; Deassign all user logicals.
UU.ZER	13	Zero a device.
UU.RAD	14	Read/Read and reset accounting data.
UU.DIR	15	Directory lookup on index; Special magnetic tape directory lookup.
UU.TRM	16	Set terminal characteristics.
UU.LOK	17	Disk directory lookup on file name; Disk wildcard directory lookup.
UU.CHE	19	Enable/disable disk caching.
UU.CNV	20	Convert date and time.
UU.SLN	21	Add/Remove/Change/List logical names.
(.MESAG)	22	Message send/receive.

Table F-1: Monitor Directives (Cont.)

MACRO Mnemonic	Function Code(FO)	Function Name
UU.SWP	23	Add/remove/list system files.
UU.JOB	24	Create a job.
UU.PPN	25	Wildcard PPN lookup.
UU.SYS	26	Return job status.
UU.PRV	28	Set/Clear/Read current privileges.
UU.STL	29	Stall/Unstall system.
UU.3PP	31	Third party privilege check.
UU.CHK	32	Check file access rights; Convert privilege name to mask; Convert privilege mask to name.
UU.ONX	33	Open next disk file.
UU.CFG	34	Set device/line printer characteristics; Set system defaults; Load/Remove monitor overlay code.
-	-	PEEK function.

Appendix G

EMT Logger Send/Receive Calls

EMT logging is an optional feature that provides a "window" on the process by which time-sharing jobs request and receive services from the RSTS/E monitor. Thus, EMT logging lets you gather information about the activity on your system. For example, you might want to know the number of logins on a particular terminal, how many files are accessed on a certain drive, or which nonresident FIP overlays get the heaviest use. Such information can help you "tune" a system for improved performance, identify bottlenecks, establish charging algorithms, and watch for potential security problems.

To use EMT logging, you must:

- Include optional code in your monitor at system installation time. See the *RSTS/E System Installation and Update Guide* for more information.

- Write a program to process the data extracted by the monitor code. This program retrieves extracted data by using send/receive calls, which are described in Chapter 8 of this manual. A demonstration program (EMTCPY.BAS) is included in the Unsupported Utility system program package. This unsupported program illustrates sample techniques for retrieving EMT logging data.

EMT logging provides information on time-sharing activity in terms of what the monitor sees. Thus, the data returned to your logging program is in terms of FIRQB and XRB contents, regardless of the language your program is written in. See the *RSTS/E System Directives Manual* for information on the FIRQB and XRB.

EMT logging can affect system performance. The impact is variable, and depends upon which EMTs you decide to log, for which jobs you log them, and how much processing your logging program attempts to do for each EMT.

Note that a feature patch is available that allows you to specify which EMTs are to be logged. See the *RSTS/E Maintenance Notebook* for details.

This appendix describes the use of parameters and other features of the send/receive calls that are specific to an EMT logger. For more information on EMT logging see the *RSTS/E System Manager's Guide*.

EMT Logging and Send/Receive

Writing an EMT logging program requires the use of several send/receive calls. The declare receiver call designates your program as a receiver and tells the monitor to activate EMT logging and build EMT data packets for selected directives. The receive local data message call retrieves EMT data packets from the monitor. The remove receiver call removes your program from the monitor's local receiver table and deactivates EMT logging.

An EMT logger program can receive messages from normal senders as well as from the monitor. (The receive call allows this selection; see Chapter 8.) Your EMT logger program should periodically check for such messages. For example, the SHUTUP program sends a message to your program before shutting down the system after normal time-sharing jobs are logged out or killed. This notification lets your program do any necessary cleanup and exit under its own control.

Declaring an EMT Logger

Your EMT logger program must declare itself as a receiver to receive messages from the monitor. The monitor recognizes an EMT logging receiver by the local object type 2 in byte 21 of the declare call; see Chapter 8.

You also need to specify the following parameters when you declare an EMT logging receiver:

- Message maximum (byte 25) - Indicates the maximum number of messages that the monitor queues for the EMT logger. (Message maximum pertains only to messages sent by other programs using normal send/receive.)

- Packet maximum (bytes 27-28) - Indicates the maximum number of EMT data packets that the monitor queues for the EMT logger. If your program cannot keep up with the data traffic and this maximum is exceeded, EMT data packets are missed (that is, not created or queued). EMT data packets may also be missed if not enough XBUF is available to hold additional

packets. Note that a count of missed EMTs is one of the parameters returned to your EMT logger program on each receive.

- Packets per message (byte 30) - Indicates the number of EMT data packets for the monitor to consider a complete message. Note that this value should not be greater than the packet maximum specified in bytes 27-28. When your program issues the receive call, the monitor immediately returns any packets that are pending, regardless of this parameter. If no packets are pending, and the receive call specifies a sleep interval, the monitor does not awaken your job until either the number of packets specified by this parameter are queued or the sleep expires. This lets your program control how often it is awakened to handle EMT packets and process more than one packet per receive. Both operations can reduce overhead.

The declare receiver call for an EMT logger has the following format. An asterisk (*) identifies fields specific to an EMT logger declare call. Other fields are more fully described in Chapter 8. Note that this form of declare receiver requires SYSIO privilege.

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; should be 0.
5-10	The receiver name.
11-20	CHR$(0%), reserved; should be 0.
21*	CHR$(2%), the object type code for an EMT logger.
22	CHR$(3%), the value for local, privileged senders (SEND privilege required).
23-24	CHR$(0%), reserved; should be 0.
25	CHR$(M%), the message maximum. This parameter pertains only to normal messages that are sent to your EMT logger.
26	CHR$(0%), the inbound link maximum; should be 0 for "no network links."

27-28*	CHR$(P.MAX%) + CHR$(SWAP%(P.MAX%)), the maximum number of EMT data packets that can be queued at one time.
29	CHR$(0%), the outbound link maximum; should be 0 for "no network links."
30*	CHR$(P.MES%), the number of packets that the monitor packs into an EMT logger message before waking up the receiver.
31-34	CHR$(0%), reserved; should be 0.
35	CHR$(R%), the receiver ID block (RIB) number.
36-40	CHR$(0%), reserved; should be 0.

Data Returned

No meaningful data is returned.

Possible Errors

See the Declare Receiver call in Chapter 8.

Receiving an EMT Logger Message

An EMT logger program asks for messages from the monitor by using a receive call in the following format. An asterisk (*) identifies fields specific to an EMT logger receive call. Other fields are more fully described in Chapter 8.

Note

The data returned to your EMT logger program depends on the internal functioning of the monitor. For this reason, both the format and meaning of data returned are subject to change in future releases of RSTS/E.

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$(4%), for L%. (N% and S% should be 0; T% has no effect for an EMT logger.)
5	CHR$(0%), to select the monitor as the sender (with byte 6).
6	CHR$(-1%), to select the monitor as the sender (with byte 5).
7-10	Reserved; should be 0.
11	CHR$(C%), the channel number for the I/O buffer to receive messages.
12	CHR$(0%), reserved; should be 0.
13-14	L%, the maximum message length (in bytes) for this receive, in the form CHR$(L%) + CHR$(SWAP%(L%)).
15-16	O%, the offset from the start of the buffer, in the form CHR$(O%) + CHR$(SWAP%(O%)).
17-26	CHR$(0%), reserved; should be 0.
27-28	T%, the sleep time in seconds, in the form CHR$(T%) + CHR$(SWAP%(T%)).
29-34	CHR$(0%), reserved; should be 0.
35	CHR$(R%), the receiver ID block (RIB) number for this receive.
36-40	CHR$(0%), reserved; should be 0.

Data Returned

Bytes	Meaning
1-2	Not meaningful; should be ignored.
3	CHR$(-1%), the local data message subfunction code.
4	CHR$(0%), the sending job (the monitor is the sender).
5-6+	0%, the sender's project-programmer number (the monitor is the sender).
7	CHR$(0%), the sender's keyboard number (the monitor is the sender).
8	Not meaningful; should be ignored.

9-10	R%, the number of bytes remaining in the data portion of the message.
11-12	Not meaningful; should be ignored.
13-14	L%, the length of the message (in bytes) transferred to the buffer.
15-20	Not meaningful; should be ignored.
21-22*	P.REM%, the number of packets remaining in the data portion of the message.
23-24*	P.EMT%, the number of EMTs missed (due to insufficient buffer space) since the last receive.
25-26*	P.TRANS%, the number of packets transferred to the buffer.
27-40	Not meaningful; should be ignored.

Possible Errors

See the receive call, in Chapter 8.

Message Format

The information returned to your EMT logger program by the receive call consists of parameters and data, as described in Chapter 8. The data portion of the message consists of packets, each of which describes a single EMT that the monitor has processed. To minimize overhead, the monitor packs as many packets as will fit into the receive buffer and returns them as a single message.

Three special parameters are returned to your EMT logger program:

- P.REM% - Indicates the number of packets pending but not yet transferred
- P.EMT% - Indicates the number of EMTs that have been missed (not logged, due to insufficient buffer space) since the last receive
- P.TRANS% - Indicates the number of packets transferred in the current receive call

These three parameters are returned in bytes 21-26 of the receive call. The monitor also returns the standard data for all receive calls; see Data Returned in the preceding section.

Each packet in the returned message is a counted string. The first two bytes of each packet contain the number of bytes in that packet, not including the two count bytes.

The rest of the packet consists of the following fields:

Root Context and control information

FIRQB Data extracted from the calling program's FIRQB

Although the identities and lengths of the fields returned are the same for all packets, you should code your EMT logger program to correctly handle the variable format; that is, honor the packet's count word and the fields' count bytes. This will make your program easier to update if the packet format changes in a future release of RSTS/E.

Figure G-1 shows the parts of an EMT packet. Each packet contains a count field as its first two bytes. This count specifies the packet's length, exclusive of the count bytes. The "parameters" returned by the Receive call contains a word (P.TRANS%) that gives the number of packets returned to your buffer.

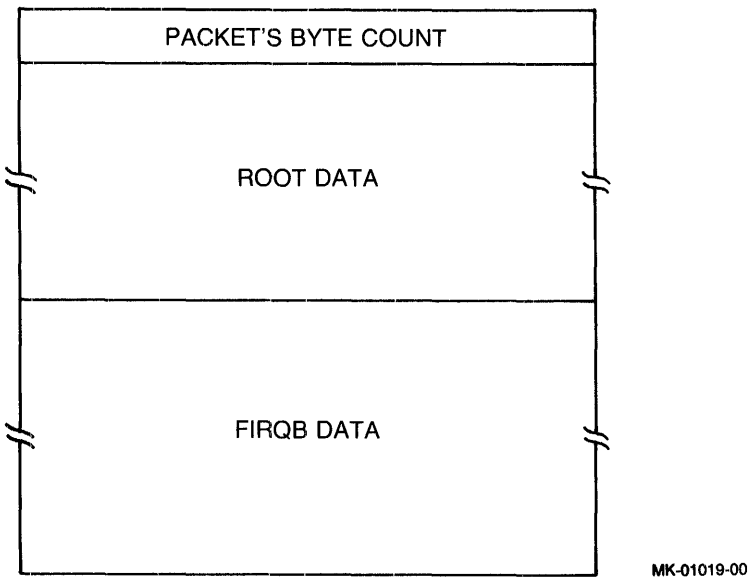


Figure G-1: EMT Data Packet Layout

EMT Root and FIRQB Fields

This section describes the EMT packet Root and FIRQB fields. The packet described here is for a single EMT. Note that a single receive may return a variable number of packets to your EMT logging program, depending on the buffer space you provide for the data portion of the message.

The EMT Root field contains the following information:

Bytes	Meaning
1-2	Reserved
3	Length of FIRQB data field
4	Length of Root data field
5-6	Packet sequence number
7-8	System date at reception of EMT
9-10	System time at reception of EMT
11	Seconds until the next minute at reception of EMT
12	Ticks until the next second at reception of EMT
13	Calling job number times two
14	Reserved
15	IOSTS byte (at directive's completion). This byte contains the returned error code. 0 indicates successful completion with no error.
16	Function code of the directive. The function codes have MACRO mnemonics of the form xxxFQ. See the *RSTS/E System Directives Manual* for a list of function codes.
17	Reserved
18	Calling job's keyboard number. In the case of a detached job, this byte is the complement of the number of the keyboard from which the job detached.
19-20	Reserved
21-22	Calling job's PPN

23-24	Job's virtual PC. This is the virtual address, in the user's job space, of the instruction following the EMT instruction that invoked the call. The PC may be of interest if the calling program is written in MACRO. The PC is within the run-time system in the case of BASIC-PLUS. For example, the PC could equal Program Counter, the PDP-11's "next instruction" register.
25	UUO code, if the call was a directive; otherwise 127. The code has a MACRO mnemonic of the form UU.xxx. See the *RSTS/E System Directives Manual* for a list of UUO codes.
26	Reserved

The EMT FIRQB field contains the following information:

Bytes	Meaning
1-2	Third word of the caller's FIRQB. The first two words of the caller's FIRQB are not returned since the same information is returned in the Root field.
3-4	Fourth word of the caller's FIRQB.
.	
.	
.	
27-28	Last word of the caller's FIRQB.

See the *RSTS/E System Directives Manual* for more information about the FIRQB.

Message from SHUTUP

As previously described, the SHUTUP system program sends a normal local data message to your EMT logger program before the system is shut down, and after normal jobs are removed. This message consists of parameters only (no data), and contains -1 in the first parameter byte (byte 21 of the returned data) as a flag.

INDEX

-A-

-E-

-M-

-N-

-Q-

-R-

-S-

-T-

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA and Puerto Rico call **800-258-1710**

In Canada call **800-267-6215**

In New Hampshire, Alaska or Hawaii call **603-884-6660**

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
100 Herzberg Road
P.O. Box 13000,
Kanata, Ontario, Canada K2K 2A6
Attn: DECDIRECT OPERATIONS

ELECTRONIC ORDERING

Dial 800-DEC-DEMO with any VT100 or VT200 compatible terminal and a 1200/2400 baud modem. If you need assistance, call 800-DEC-INFO.

INTERNATIONAL

DIGITAL EQUIPMENT CORPORATION
P&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473-0471

*Any prepaid order from Puerto Rico must be placed with the Local Digital Subsidiary:
809-754-7575

RSTS/E
Programming Manual
AA-EZ09A-TC
Including AD-EZ09A-T1,T2

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: Office Systems Documentation MK01-2/E02
DIGITAL EQUIPMENT CORPORATION
Continental Boulevard
Merrimack N.H. 03054

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line