

**August 1978**

This document contains the information necessary to interface directly with the I/O device drivers supplied as part of the VAX/VMS operating system. Several examples of programming techniques are included. This document does not contain information on I/O operations using VAX-11 Record Management Services.

## **VAX/VMS I/O User's Guide**

Order No. AA-D028A-TE

<b>SUPERSESSION/UPDATE INFORMATION:</b>	This is a new document for this release.
<b>OPERATING SYSTEM AND VERSION:</b>	VAX/VMS V01
<b>SOFTWARE VERSION:</b>	VAX/VMS V01

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

**digital equipment corporation • maynard, massachusetts**

First Printing, August 1978

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 only be used or copied 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.

Copyright © 1978 by Digital Equipment Corporation

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECTape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	

## CONTENTS

		Page
PREFACE		ix
CHAPTER 1	INTRODUCTION TO VAX/VMS INPUT/OUTPUT	1-1
1.1	OVERVIEW OF VAX/VMS I/O	1-1
1.2	VAX/VMS I/O DEVICES	1-1
1.3	SUMMARY OF I/O SYSTEM SERVICES	1-2
1.4	QUOTAS, PRIVILEGES, AND PROTECTION	1-3
1.4.1	Buffered I/O Quota	1-3
1.4.2	Buffered I/O Byte Count Quota	1-4
1.4.3	Direct I/O Quota	1-4
1.4.4	AST Quota	1-4
1.4.5	Physical I/O Privilege (PHY_IO)	1-4
1.4.6	Logical I/O Privilege (LOG_IO)	1-4
1.4.7	Mount Privilege	1-5
1.4.8	Volume Protection	1-5
1.5	SUMMARY OF VAX/VMS QIO OPERATIONS	1-6
1.6	PHYSICAL, LOGICAL, AND VIRTUAL I/O	1-6
1.6.1	Physical I/O Operations	1-6
1.6.2	Logical I/O Operations	1-8
1.6.3	Virtual I/O Operations	1-10
1.7	I/O FUNCTION ENCODING	1-12
1.7.1	Function Codes	1-12
1.7.2	Function Modifiers	1-12
1.8	ISSUING I/O REQUESTS	1-13
1.8.1	Channel Assignments	1-13
1.8.2	Device Allocation	1-14
1.8.3	I/O Function Requests	1-14
1.8.4	\$QIO Macro Format	1-15
1.8.5	\$QIOW Macro Format	1-16
1.8.6	\$QIO and \$QIOW Arguments	1-16
1.8.6.1	Event Flag Number Argument	1-17
1.8.6.2	Channel Number Argument	1-17
1.8.6.3	Function Argument	1-17
1.8.6.4	I/O Status Block Argument	1-18
1.8.6.5	AST Address Argument	1-18
1.8.6.6	AST Parameter Argument	1-18
1.8.6.7	Device/Function Dependent Arguments	1-18
1.8.7	\$INPUT and \$OUTPUT Macro Format and Arguments	1-19
1.8.8	Status Returns for System Services	1-19
1.9	I/O COMPLETION	1-21
1.9.1	Event Flags	1-22
1.9.2	I/O Status Block	1-22
1.9.3	Asynchronous System Traps	1-23
1.10	DEVICE INFORMATION	1-23
1.10.1	\$GETCHN and \$GETDEV Status Returns	1-25
CHAPTER 2	TERMINAL DRIVER	2-1
2.1	SUPPORTED TERMINAL DEVICES	2-1
2.2	TERMINAL DRIVER FEATURES AND CAPABILITIES	2-1
2.2.1	Type-ahead	2-1
2.2.2	Line Terminators	2-2
2.2.3	Special Operating Modes	2-2
2.2.4	Escape Sequences	2-2
2.2.5	Terminal/Mailbox Interaction	2-3
2.2.6	Control Characters and Special Keys	2-5

## CONTENTS (Cont.)

	Page
2.2.6.1	2-8
2.2.7	2-10
2.3	2-10
2.4	2-13
2.4.1	2-14
2.4.1.1	2-15
2.4.1.2	2-16
2.4.2	2-16
2.4.2.1	2-17
2.4.2.2	2-17
2.4.3	2-20
2.4.3.1	2-23
2.4.3.2	2-23
2.5	2-24
2.6	2-26
CHAPTER 3	3-1
3.1	3-1
3.1.1	3-1
3.1.2	3-1
3.1.3	3-2
3.2	3-2
3.2.1	3-3
3.2.2	3-3
3.2.3	3-4
3.3	3-6
3.4	3-9
3.4.1	3-10
3.4.2	3-10
3.4.3	3-10
3.4.3.1	3-11
3.4.3.2	3-12
3.5	3-14
3.6	3-14
CHAPTER 4	4-1
4.1	4-1
4.1.1	4-1
4.2	4-2
4.2.1	4-2
4.2.2	4-3
4.2.3	4-3
4.3	4-5
4.4	4-8
4.4.1	4-10
4.4.2	4-10
4.4.3	4-11
4.4.4	4-11
4.4.5	4-11
4.4.6	4-12
4.4.7	4-12
4.4.8	4-12
4.4.9	4-12
4.4.9.1	4-12
4.4.9.2	4-13

## CONTENTS (Cont.)

		Page
4.5	I/O STATUS BLOCK	4-14
4.6	PROGRAMMING EXAMPLE	4-17
CHAPTER 5	LINE PRINTER DRIVER	5-1
5.1	SUPPORTED LINE PRINTER DEVICES	5-1
5.1.1	LP11 Line Printer Interface	5-1
5.1.2	L111 DECprinter I	5-1
5.2	DRIVER FEATURES AND CAPABILITIES	5-1
5.2.1	Output Character Formatting	5-2
5.2.2	Error Recovery	5-2
5.3	DEVICE INFORMATION	5-3
5.4	LINE PRINTER FUNCTION CODES	5-4
5.4.1	Write	5-4
5.4.1.1	Write Function Carriage Control	5-5
5.4.2	Sense Printer Mode	5-8
5.4.3	Set Mode	5-8
5.5	I/O STATUS BLOCK	5-9
5.6	PROGRAMMING EXAMPLE	5-10
CHAPTER 6	CARD READER DRIVER	6-1
6.1	SUPPORTED CARD READER DEVICE	6-1
6.2	DRIVER FEATURES AND CAPABILITIES	6-1
6.2.1	Read Modes	6-1
6.2.2	Special Card Punch Combinations	6-2
6.2.2.1	End-of-File Condition	6-2
6.2.2.2	Set Translation Mode	6-2
6.2.3	Error Recovery	6-2
6.3	DEVICE INFORMATION	6-3
6.4	CARD READER FUNCTION CODES	6-5
6.4.1	Read	6-6
6.4.2	Sense Card Reader Mode	6-7
6.4.3	Set Mode	6-7
6.4.3.1	Set Mode	6-7
6.4.3.2	Set Characteristic	6-8
6.5	I/O STATUS BLOCK	6-8
CHAPTER 7	MAILBOX DRIVER	7-1
7.1	MAILBOX OPERATIONS	7-1
7.1.1	Creating Mailboxes	7-2
7.1.2	Deleting Mailboxes	7-3
7.1.3	Mailbox Message Format	7-3
7.2	DEVICE INFORMATION	7-4
7.3	MAILBOX FUNCTION CODES	7-5
7.3.1	Read	7-5
7.3.2	Write	7-6
7.3.3	Write End-of-File Message	7-7
7.3.4	Set Attention AST	7-7
7.4	I/O STATUS BLOCK	7-9
7.5	PROGRAMMING EXAMPLE	7-10
CHAPTER 8	DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER	8-1
8.1	SUPPORTED DMC11 SYNCHRONOUS LINE INTERFACES	8-1

## CONTENTS (Cont.)

	Page
8.1.1 DIGITAL Data Communications Message Protocol	8-1
8.2 DRIVER FEATURES AND CAPABILITIES	8-2
8.2.1 Mailbox Usage	8-2
8.2.2 Quotas	8-3
8.2.3 Power Failure	8-3
8.3 DEVICE INFORMATION	8-3
8.4 DMC11 FUNCTION CODES	8-6
8.4.1 Read	8-6
8.4.2 Write	8-7
8.4.3 Set Mode	8-7
8.4.3.1 Set Mode and Set Characteristics	8-8
8.4.3.2 Enable Attention AST	8-8
8.4.3.3 Set Mode and Shut Down Unit	8-9
8.4.3.4 Set Mode and Start Unit	8-9
8.5 I/O STATUS BLOCK	8-10
 CHAPTER 9 QIO INTERFACE TO FILE SYSTEM ACPs	 9-1
9.1 ACP FUNCTIONS AND ENCODING	9-1
9.1.1 ACP Device/Function-Dependent Arguments	9-2
9.2 FILE INFORMATION BLOCK	9-3
9.3 ATTRIBUTE CONTROL BLOCK	9-11
9.4 I/O STATUS BLOCK	9-14
 APPENDIX A DISK, MAGNETIC TAPE AND ACP QIO FUNCTIONS	 A-1
 APPENDIX B I/O FUNCTION CODES	 B-1

## FIGURES

FIGURE	1-1 Physical I/O Access Checks	1-7
	1-2 Logical I/O Access Checks	1-9
	1-3 Physical, Logical, and Virtual I/O	1-11
	1-4 I/O Function Format	1-12
	1-5 Function Modifier Format	1-13
	1-6 System Service Status Return	1-20
	1-7 I/O Status Block Format	1-22
	1-8 CALL Instruction Argument List	1-23
	1-9 Buffer Format for \$GETCHN and \$GETDEV System Services	1-25
	2-1 Terminal Mailbox Message Format	2-4
	2-2 Character Interpretation	2-9
	2-3 Terminal Information	2-10
	2-4 Short and Long Forms of Terminator Mask Quadwords	2-16
	2-5 P4 Carriage Control Specifier	2-17
	2-6 Write Function Carriage Control (Prefix and Postfix Coding)	2-20
	2-7 Set Mode Characteristics Buffer	2-21
	2-8 IOSB Contents - Read Function	2-24
	2-9 IOSB Contents - Write Function	2-24
	2-10 IOSB Contents - Set Mode and Set Characteristic Functions	2-25
	3-1 Disk Information	3-4
	3-2 Starting Physical Address	3-9
	3-3 Set Mode Characteristics Buffer	3-11

CONTENTS (Cont.)

		Page
FIGURES (Cont.)		
FIGURE	3-4 Set Characteristic Buffer	3-11
	3-5 IOSB Content	3-12
	4-1 Magnetic Tape Information	4-3
	4-2 IO\$ SKIPFILE Argument	4-11
	4-3 IO\$ SKIPRECORD Argument	4-11
	4-4 Set Mode Characteristics Buffer	4-12
	4-5 Set Characteristic Buffer	4-13
	4-6 IOSB Content	4-14
	5-1 Printer Information	5-3
	5-2 P4 Carriage Control Specifier	5-5
	5-3 Write Function Carriage Control (Prefix and Postfix Coding)	5-7
	5-4 Set Mode Characteristics Buffer	5-8
	5-5 Set Characteristics Characteristics Buffer	5-8
	5-6 IOSB Contents - Write Function	5-9
	5-7 IOSB Contents - Set Mode Function	5-9
	6-1 Card Reader Information	6-4
	6-2 Binary and Packed Column Storage	6-6
	6-3 Set Mode Characteristics Buffer	6-7
	6-4 Set Characteristic Buffer	6-8
	6-5 IOSB Contents	6-9
	7-1 Multiple Mailbox Channels	7-3
	7-2 Typical Mailbox Message Format	7-4
	7-3 Mailbox Information	7-4
	7-4 Read Mailbox	7-6
	7-5 Write Mailbox	7-7
	7-6 Write Attention AST (Read Unsolicited Data)	7-8
	7-7 Read Attention AST	7-9
	7-8 IOSB Contents - Read Function	7-9
	7-9 IOSB Contents - Write Function	7-10
	8-1 Mailbox Message Format	8-3
	8-2 DMCI1 Information	8-3
	8-3 P1 Characteristics Block	8-8
	8-4 IOSB Content	8-10
	9-1 ACP QIO Interface	9-1
	9-2 ACP Device/Function-Dependent Arguments	9-2
	9-3 ACP Device/Function Argument Descriptor Format	9-3
	9-4 File Information Block Format	9-3
	9-5 Typical Short File Information Block	9-4
	9-6 Attribute Control Block Format	9-12
	9-7 IOSB Contents - ACP QIO Functions	9-14

TABLES

TABLE	1-1 Read and Write I/O Functions	1-12
	1-2 Device/Function Independent Arguments	1-16
	1-3 \$INPUT and \$OUTPUT Arguments	1-19
	1-4 \$QIO, \$QIOW, \$INPUT, and \$OUTPUT System Services Status Returns	1-20
	1-5 \$GETCHN and \$GETDEV Arguments	1-24
	1-6 \$GETCHN and \$GETDEV Status Returns	1-26

## CONTENTS (Cont.)

		Page
TABLES (Cont.)		
TABLE	2-1 Terminal Control Characters	2-5
	2-2 Special Terminal Keys	2-8
	2-3 Terminal Device-Independent Characteristics	2-11
	2-4 Terminal Characteristics	2-12
	2-5 Read QIO Function Modifiers	2-15
	2-6 Write QIO Function Modifiers	2-17
	2-7 Write Function Carriage Control (FORTRAN: Byte 0 not equal to 0)	2-18
	2-8 Write Function Carriage Control (P4 byte 0 = 0)	2-19
	2-9 Value for Set Mode and Set Characteristic P1 Characteristics	2-22
	2-10 Terminal QIO Status Returns	2-25
	3-1 Disk Devices	3-1
	3-2 Disk Device Characteristics	3-5
	3-3 Disk I/O Functions	3-7
	3-4 Status Returns for Disk Devices	3-12
	4-1 Magnetic Tape Devices	4-1
	4-2 Magnetic Tape Device-Independent Characteristics	4-4
	4-3 Device-Dependent Information for Tape Devices	4-4
	4-4 Magnetic Tape I/O Functions	4-6
	4-5 Set Mode and Set Characteristic Magnetic Tape Characteristics	4-13
	4-6 Status Returns for Tape Devices	4-14
	5-1 Printer Device-Independent Characteristics	5-3
	5-2 Printer Device-Dependent Characteristics	5-4
	5-3 Write Function Carriage Control (FORTRAN: Byte 0 not equal to 0)	5-6
	5-4 Write Function Carriage Control (P4 byte 0 equal to 0)	5-6
	5-5 Line Printer QIO Status Returns	5-9
	6-1 Card Reader Device-Independent Characteristics	6-4
	6-2 Device-Dependent Information for Card Readers	6-5
	6-3 Card Reader I/O Functions	6-5
	6-4 Set Mode and Set Characteristic Card Reader Characteristics	6-8
	6-5 Status Returns for Card Reader	6-9
	7-1 Mailbox Read and Write Operations	7-1
	7-2 Mailbox Characteristics	7-4
	7-3 Mailbox QIO Status Returns	7-10
	8-1 Supported DMC11 Options	8-1
	8-2 DMC11 Device Characteristics	8-4
	8-3 DMC11 Device Types	8-4
	8-4 DMC11 Unit Characteristics	8-5
	8-5 DMC11 Unit and Line Status	8-5
	8-6 Error Summary Bits	8-6
	8-7 Status Returns for DMC11	8-11
	9-1 Contents of the File Information Block	9-5
	9-2 FIB Argument Usage in ACP QIO Functions	9-9
	9-3 Attribute Control Block Fields	9-12
	9-4 ACP QIO Attributes	9-13
	9-5 ACP QIO Status Returns	9-15
	A-1 IO\$_CREATE Arguments	A-2
	A-2 IO\$_ACCESS Arguments	A-4
	A-3 IO\$_MODIFY Arguments	A-6

## PREFACE

### MANUAL OBJECTIVES

This manual provides users of the VAX/VMS operating system with the information necessary to interface directly with the I/O device drivers supplied as part of the operating system. It is not the objective of this manual to provide the reader with information on all aspects of VAX/VMS input/output (I/O) operations.

### INTENDED AUDIENCE

This manual is intended for system programmers who want to take advantage of the time and/or space savings that result from direct use of the I/O device drivers. Readers are expected to have some experience with either VAX-11 FORTRAN IV-PLUS or VAX-11 MACRO assembly language. Users of VAX/VMS who do not require such detailed knowledge of I/O drivers can use the device-independent services described in the VAX-11 Record Management Services Reference Manual.

### STRUCTURE OF THIS DOCUMENT

This manual is organized into nine chapters and two appendixes, as follows:

- Chapter 1 contains introductory information. It provides an overview of VAX/VMS I/O operations; I/O system services; and I/O quotas, privileges, and protection. This chapter also introduces I/O function encoding and how to make I/O requests, and describes how to obtain information on the different devices.
- Chapters 2 through 8 describe the use of all the I/O device drivers supported by VAX/VMS:
  - Chapter 2 deals with the terminal driver
  - Chapter 3 deals with disk drivers
  - Chapter 4 deals with magnetic tape drivers
  - Chapter 5 deals with the line printer driver
  - Chapter 6 deals with the card reader driver
  - Chapter 7 deals with the mailbox driver
  - Chapter 8 deals with the DMCl1 driver

- Chapter 9 describes the Queue I/O (QIO) interface to file system ancillary control processes (ACPs).
- Appendix A describes the QIO functions that are common to the disk and magnetic tape drivers and the ACP QIO interface.
- Appendix B summarizes the QIO function codes, arguments, and function modifiers used by the different device drivers.

#### ASSOCIATED DOCUMENTS

The following documents may also be useful:

- VAX-11 Information Directory - contains a complete list of all VAX-11 documents
- VAX/VMS System Services Reference Manual
- VAX-11 Linker Reference Manual
- VAX-11 Software Handbook
- PDP-11 Peripherals Handbook
- VAX-11 FORTRAN IV-PLUS User's Guide
- VAX-11 MACRO User's Guide
- VAX-11 Record Management Services Reference Manual

#### CONVENTIONS USED IN THIS MANUAL

The following conventions are used in this manual:

- Brackets ([]) in QIO requests enclose optional arguments. For example:

```
IO$_CREATE P1,[P2],[P3],[P4],[P5]
```

- Horizontal ellipses (...) indicate that characters or QIO arguments that are not pertinent to the example have been omitted. For example:

```
(that is, 8, 16, 24,...).
```

- Vertical ellipses in coding examples indicate that lines of code not pertinent to the example are omitted. For example:

```
TTCHAN: .BLKW 1
        .
        .
        $ASSIGN_S DEVNAM=TTNAME,CHAN=TTCHAN
```

- Hyphens (-) in coding examples indicate that additional arguments to the QIO request are provided on the following line(s). For example:

```

$QIO_S  FUNC=#IO$_WRITEPBLK,-           ;FUNCTION IS
-                                             ;WRITE PHYSICAL
        CHAN=W^TTCHAN1,-               ;TO TTCHAN 1
        EFN=#1,-                       ;EVENT FLAG 1
        P1=W^ASTMSG,-                  ;P1 = BUFFER
        P2=#ASTMSGSIZE                 ;P2 = BUFFER SIZE

```

- Angle brackets (<>) enclose a hexadecimal number representing an ASCII character code or a mnemonic for an ASCII character on the terminal keyboard. For example:

```

(ESC) <0> <20-2F>...<40-7E>

```

- Unless otherwise noted, all numbers in the text are assumed to be decimal. In coding examples, the radix -- binary, octal, decimal, or hexadecimal -- will be explicitly indicated.



## CHAPTER 1

### INTRODUCTION TO VAX/VMS INPUT/OUTPUT

VAX/VMS supports a variety of input and output (I/O) devices, including disks, terminals, magnetic tapes, card readers, line printers, synchronous line interfaces, and software mailboxes. This manual describes the capabilities of VAX/VMS device drivers and their programming interface and gives several simple programming examples that use I/O drivers to perform input/output operations.

#### 1.1 OVERVIEW OF VAX/VMS I/O

Input/output operations under VAX/VMS are designed to be as device- and function-independent as possible. User processes issue I/O requests to software channels which form paths of communication with a particular device. Each process can establish its own correspondence between physical devices and channels. I/O requests are queued when they are issued, and processed according to the relative priority of the process that issued them. I/O requests can be handled indirectly by the VAX-11 Record Management Services (RMS) or they can interface directly to the VAX/VMS I/O system. (VAX-11 RMS is described in the VAX-11 Record Management Services Reference Manual.)

To access the I/O services described in this manual, users issue system service requests. In certain system service requests, a function code included in the request defines the particular operation to be performed. For example, Queue I/O (QIO) system service requests can specify such operations as reading and writing blocks of data.

QIO requests can also specify a number of device-specific input/output operations; for example, converting lowercase characters to uppercase in terminal read operations, and rewinding magnetic tape.

#### 1.2 VAX/VMS I/O DEVICES

VAX/VMS supports the following devices:

- Terminals, using the DZ11 Asynchronous Serial Line Multiplexer, and the VAX-11/780 console
- Disk devices:
  - RM03 Pack Disk
  - RP05 and RP06 Pack Disks
  - RK06 and RK07 Cartridge Disks

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

- TE16 Magnetic Tape
- Line printers:
  - LP11 Line Printer Interface
  - LA11 DECprinter
- CR11 Card Reader
- DMC11 Synchronous Line Interface
- Mailboxes -- virtual devices used for interprocess transfer of information

Chapters 2 through 8 describe in detail the drivers for these I/O devices and the I/O operations they perform.

### 1.3 SUMMARY OF I/O SYSTEM SERVICES

The following system services allow the direct use of the operating system's I/O resources:

- Assign I/O Channel (\$ASSIGN) system service
- Deassign I/O Channel (\$DASSGN) system service
- Queue I/O Request (\$QIO) system service
- Queue I/O Request and Wait for Event Flag (\$QIOW) system service
- Allocate Device (\$ALLOC) system service
- Deallocate Device (\$DALLOC) system service
- Get Channel Information (\$GETCHN) system service
- Get Device Information (\$GETDEV) system service
- Cancel I/O on Channel (\$CANCEL) system service
- Create Mailbox and Assign Channel (\$CREMBX) system service
- Delete Mailbox (\$DELMBX) system service
- Wait for Single Event Flag (\$WAITFR) system service
- Wait for Logical AND of Event Flags (\$WFLAND) system service
- Wait for Logical OR of Event Flags (\$WFLOR) system service
- Set AST Enable (\$SETAST) system service
- Set Resource Wait Mode (\$SETRWM) system service

This manual describes the use of system services for I/O operations. It also describes other system services used with I/O operations such as asynchronous system traps (ASTs) and event flag services. Section 1.8 describes the QIO request system service; ASTs and event flags, and \$GETCHN are described in Sections 1.9 and 1.10, respectively. Section 1.8.7 describes the use of the \$INPUT and \$OUTPUT macros, which perform functions similar to the \$QIOW macro.

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

See the VAX/VMS System Services Reference Manual for detailed information on all these system services and examples of their use. The VAX/VMS System Services Reference Manual also contains information on physical and logical device-naming conventions.

### 1.4 QUOTAS, PRIVILEGES, AND PROTECTION

To preserve the integrity of the system, VAX/VMS I/O operations are performed under the constraints of quotas, privileges, and protection.

Quotas establish a limit on the number and type of I/O operations that a process can perform concurrently. They ensure that all users have an equitable share of system resources and usage.

Privileges are granted to a user to allow the performance of certain I/O-related operations; for example, create a mailbox and perform logical I/O to a file-structured device. Restrictions on user privilege protect the integrity and performance of both the operating system and the services provided other users.

Protection is used to control access to files and devices. Device protection is provided in much the same way as file protection: shareable and nonshareable file devices and shareable nonfile devices such as mailboxes, are protected by protection masks. Nonshareable, nonfile devices such as terminals, can be accessed if they are not allocated to another process.

The Set Resource Wait Mode (\$SETRWM) system service allows a process to select either of two modes when an attempt to exceed a quota occurs. In the enabled (default) mode, the process waits until the required resource is available before continuing. In the disabled mode, the process is notified immediately by a system service status return that an attempt to exceed a quota has occurred. Waiting for resources is transparent to the process when resource wait mode is enabled; no explicit action is taken by the process when a wait is necessary.

The different types of I/O-related quotas, privileges, and protection are described in the following paragraphs.

#### 1.4.1 Buffered I/O Quota

The buffered I/O quota specifies the maximum number of concurrent buffered I/O operations a process can have active. In a buffered I/O operation, the user's data is buffered in system dynamic memory. The driver deals with the system buffer and not the user buffer. Buffered I/O is used for terminal, line printer, card reader, and mailbox transfers. The user's buffer does not have to be locked in memory for a buffered I/O operation.

The buffered I/O quota value is established in the user authorization file by the system manager or by the process's creator. Resource wait mode is entered if enabled by the Set Resource Wait Mode system service and an attempt to exceed the buffered I/O quota is made.

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

### 1.4.2 Buffered I/O Byte Count Quota

The buffered I/O byte count quota specifies the maximum amount of buffer space that can be concurrently consumed from system dynamic memory for buffering I/O requests. All buffered I/O requests require system dynamic memory in which the actual I/O operation takes place.

The buffered I/O byte count quota is established in the user authorization file by the system manager or by the process's creator. Resource wait mode is entered if enabled by the Set Resource Wait Mode system service and an attempt to exceed the buffered I/O byte count quota is made.

### 1.4.3 Direct I/O Quota

The direct I/O quota specifies the maximum number of concurrent direct, that is, unbuffered, I/O operations that a process can have active. In a direct I/O operation, data is moved directly to or from the user buffer. Direct I/O is used for disk, magnetic tape, and DMC11 transfers. For direct I/O, the user's buffer must be locked in memory during the transfer.

The direct I/O quota value is established in the user authorization file by the system manager or by the process's creator. Resource wait mode is entered if enabled by the Set Resource Wait Mode system service and an attempt to exceed the direct I/O quota is made.

### 1.4.4 AST Quota

The AST quota specifies the maximum number of asynchronous system traps that a process can have outstanding. The quota value is established in the user authorization file by the system manager or by the process's creator. There is never an implied wait for this resource.

### 1.4.5 Physical I/O Privilege (PHY\_IO)

Physical I/O privilege allows a process to perform physical I/O operations on a device. Physical I/O privilege also allows a process to perform logical I/O operations on a device. (Figures 1-1 and 1-2 show the use of physical I/O privilege in greater detail.)

### 1.4.6 Logical I/O Privilege (LOG\_IO)

Logical I/O privilege allows a process to perform logical I/O operations on a device. A process can also perform physical operations on a device if the process has logical I/O privilege, the volume is mounted foreign, and the volume protection mask allows access to the device. (Figures 1-1 and 1-2 show the use of logical I/O privilege in greater detail.)

# INTRODUCTION TO VAX/VMS INPUT/OUTPUT

## 1.4.7 Mount Privilege

Mount privilege allows a process to use the IO\$\_MOUNT function to perform mount operations on disk and magnetic tape devices. IO\$\_MOUNT is used in ACP interface operations (see Chapter 9).

## 1.4.8 Volume Protection

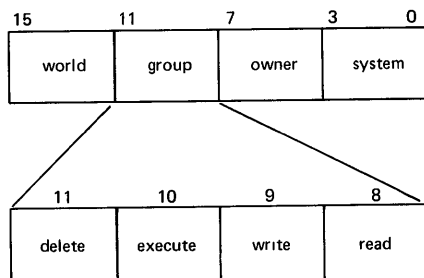
Volume protection protects the integrity of mailboxes and both foreign and Files-11 structured volumes. Volume protection for a foreign volume is established when the volume is mounted. Volume protection for a Files-11 structured volume is established when the volume is initialized. (The protection can be overridden when the volume is mounted if the process that is mounting the volume has the override volume protection privilege.)

Mailbox protection is established by the \$CREMBX system service protection mask argument.

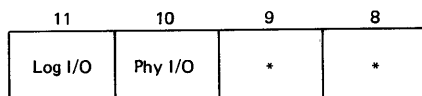
Protection for structured volumes and mailboxes is provided by a volume protection mask that contains four 4-bit fields. These fields correspond to the four classes of users that are permitted to access the volume. (User classes are based on the volume owner's user identification code (UIC).)

The 4-bit fields are interpreted differently for volumes that are mounted as structured (that is, volumes serviced by an Ancillary Control Process ACP) and volumes that are mounted as foreign.

The 4-bit fields have the following format for volumes mounted as structured:



The 4-bit fields have the following format for volumes mounted as foreign:



\*not used

Usually, volume protection is meaningful only for read and write operations.

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

### 1.5 SUMMARY OF VAX/VMS QIO OPERATIONS

VAX/VMS provides QIO operations that perform three basic I/O functions: read, write, and set mode. The read function transfers data from a device to a user-specified buffer. The write function transfers data in the opposite direction - from a user-specified buffer to the device. For example, in a read QIO function to a terminal device, a user-specified buffer is filled with characters received from the terminal. In a write QIO function to the terminal, the data in a user-specified buffer is transferred to the terminal where it is displayed.

The set mode QIO function is used to control or describe the characteristics and operation of a device. For example, a set mode QIO function to a line printer can specify either uppercase or lowercase character format. Not all QIO functions are applicable to all types of devices. The line printer, for example, cannot perform a read QIO function.

### 1.6 PHYSICAL, LOGICAL, AND VIRTUAL I/O

I/O data transfers can occur in any one of three device addressing modes: physical, logical, or virtual. Any process with device access allowed by the volume protection mask can perform logical I/O on a device that is mounted foreign; physical I/O requires privilege. Virtual I/O does not require privilege; however, intervention by an ACP to control user access may be necessary if the device is under ACP control. (ACP functions are described in Chapter 9.)

#### 1.6.1 Physical I/O Operations

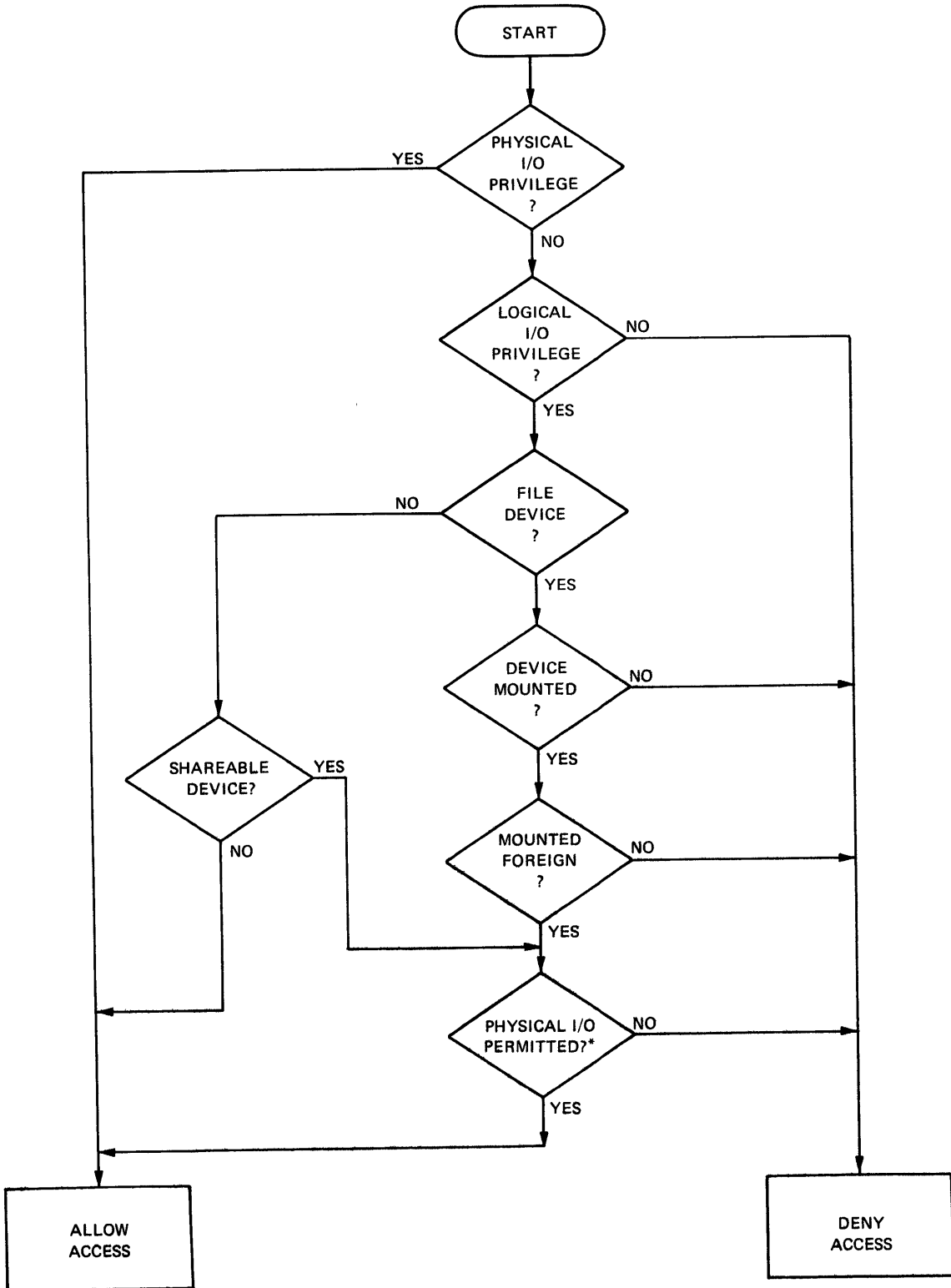
In physical I/O operations, data is read from and written to the actual, physically addressable units accepted by the hardware; for example, sectors on a disk or binary characters on a terminal in the PASSALL mode. This mode allows direct access to all device-level I/O operations.

Physical I/O requires that one of the following conditions be met:

- The issuing process has physical I/O privilege (PHY\_IO)
- The issuing process has logical I/O privilege (LOG\_IO), the device is mounted foreign, and the volume protection mask allows physical access to the device

If neither of these conditions is met, the physical I/O operation is rejected by the QIO system service with a status return of SSS\_NOPRIV (no privilege). Figure 1-1 illustrates the physical I/O access checks in greater detail.

INTRODUCTION TO VAX/VMS INPUT/OUTPUT



\*Volume protection mask allows access

Figure 1-1 Physical I/O Access Checks

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

The inhibit error logging function modifier (IO\$M INHERLOG) can be specified for all physical I/O functions. IO\$M INHERLOG inhibits the logging of any error that occurs during the I/O operation.

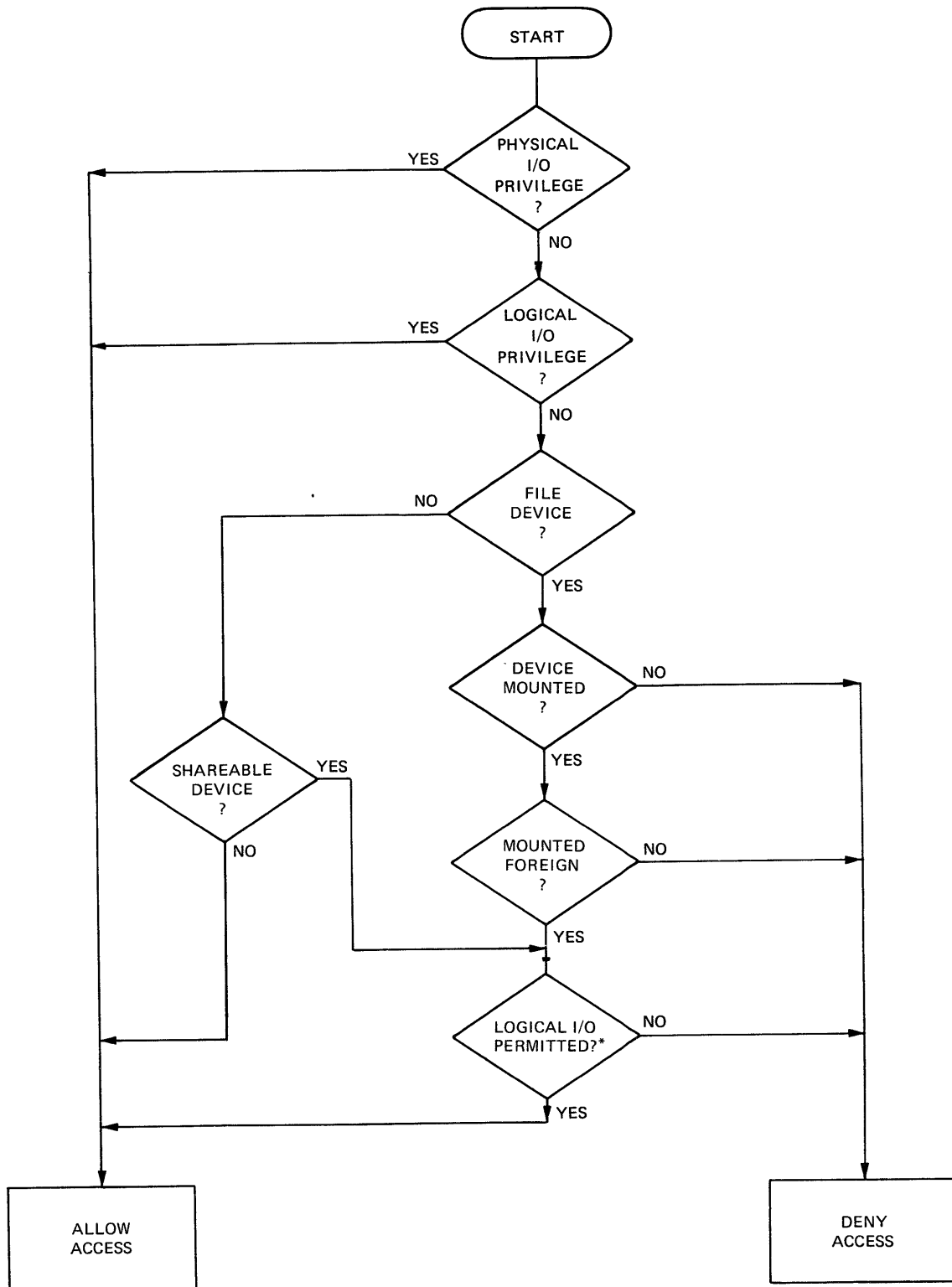
### 1.6.2 Logical I/O Operations

In logical I/O operations, data is read from and written to logically addressable units of the device. Logical operations can be performed on both block-addressable and record-oriented devices. For block-addressable devices (for example, disks), the addressable units are 512-byte blocks. They are numbered from 0 to n where n is the last block on the device. For record-oriented or non-block-structured devices (for example, terminals), logical addressable units are not pertinent and are ignored. Logical I/O requires that one of the following conditions be met:

- The issuing process has physical I/O privilege (PHY\_IO)
- The issuing process has logical I/O privilege (LOG\_IO)
- The volume is mounted foreign and the volume protection mask allows access to the device

If none of these conditions is met, the logical I/O operation is rejected by the QIO system service with a status return of SSS\_NOPRIV (no privilege). Figure 1-2 illustrates the logical I/O access checks in greater detail.

# INTRODUCTION TO VAX/VMS INPUT/OUTPUT



\*Volume protection mask allows access

Figure 1-2 Logical I/O Access Checks

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

### 1.6.3 Virtual I/O Operations

Virtual I/O operations can be performed on both record-oriented (non-file-structured) and block-addressable (file-structured) devices. For record-oriented devices (for example, terminals), the virtual function is the same as a logical function; the virtual addressable units of the devices are ignored.

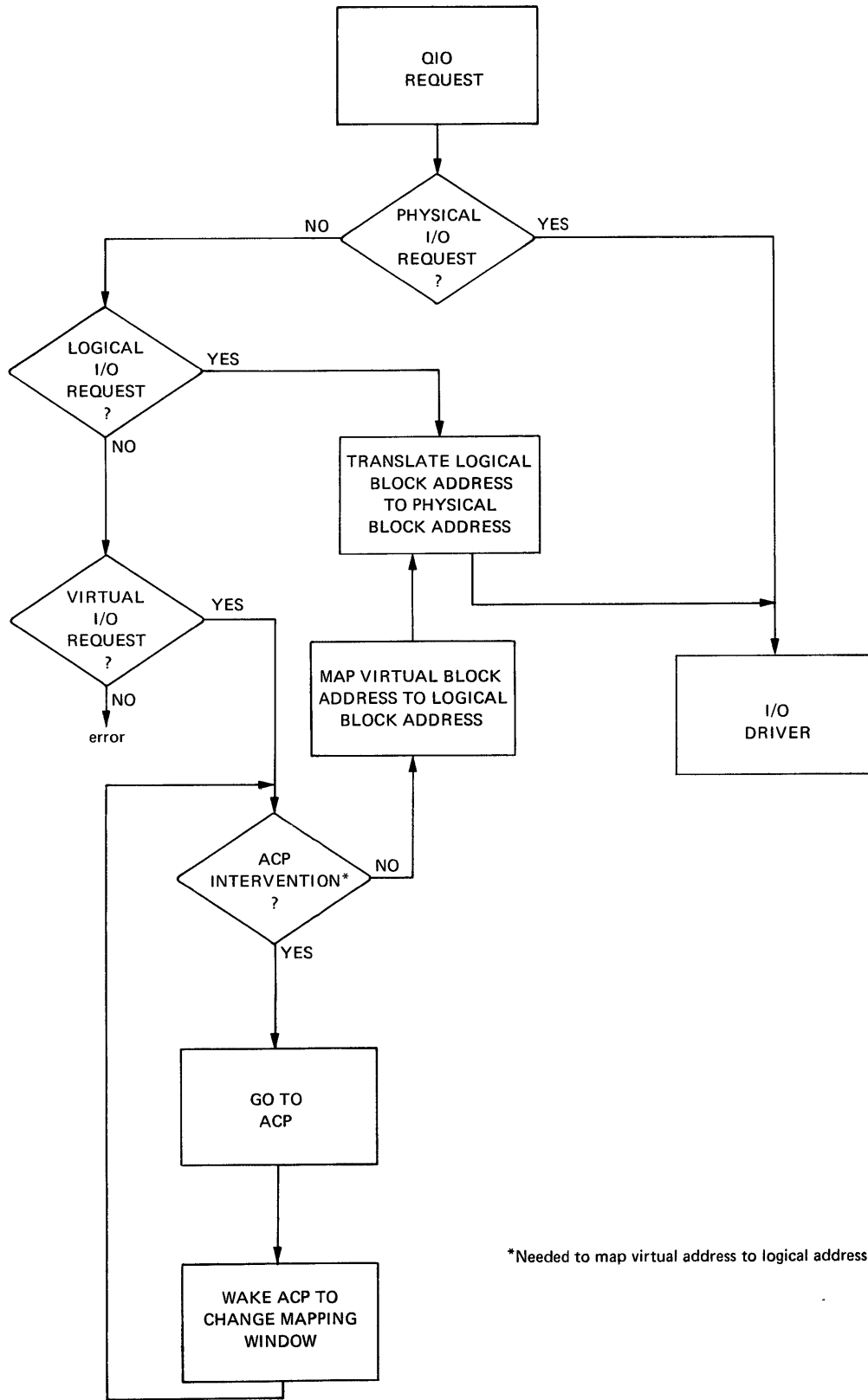
For block-addressable devices (for example, disks), data is read from and written to open files. The addressable units in the file are 512-byte blocks. They are numbered starting at 1 and are relative to a file rather than to a device. Block-addressable devices must be mounted, structured, and contain a previously opened file.

Virtual I/O operations also require that the volume protection mask allow access to the device (a process having either physical or logical I/O privilege can override the volume protection mask). If these conditions are not met, the virtual I/O operation is rejected by the QIO system service with one of the following status returns:

<u>Status Return</u>	<u>Meaning</u>
SS\$_NOPRIV	No privilege
SS\$_DEVNOTMOUNT	Device not mounted
SS\$_DEVFOREIGN	Volume mounted foreign (a foreign volume is a volume that does not contain a standard file structure understood by any of the VAX/VMS software)

Figure 1-3 shows the relationship of physical, logical, and virtual I/O to the driver.

INTRODUCTION TO VAX/VMS INPUT/OUTPUT



\*Needed to map virtual address to logical address

Figure 1-3 Physical, Logical, and Virtual I/O

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

### 1.7 I/O FUNCTION ENCODING

I/O functions fall into three groups that correspond to the three I/O device addressing modes (physical, logical, and virtual) described in Section 1.6. Depending on the device to which it is directed, an I/O function can be expressed in one, two, or all three modes.

I/O functions are described by 16-bit, symbolically-expressed values that specify the particular I/O operation to be performed and any optional function modifiers. Figure 1-4 shows the format of the 16-bit function value.

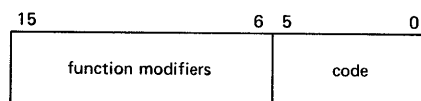


Figure 1-4 I/O Function Format

Symbolic names for I/O function codes are defined by the \$IODEF macro, as described in the VAX/VMS System Services Reference Manual.

#### 1.7.1 Function Codes

The low-order 6 bits of the function value are a code that specifies the particular operation to be performed. For example, the code for read logical block is expressed as IO\$\_READLBLK. Table 1-1 lists the symbolic values for read and write I/O functions in the three transfer modes.

Table 1-1  
Read and Write I/O Functions

Physical I/O	Logical I/O	Virtual I/O
IO\$_READPBLK IO\$_WRITEPBLK	IO\$_READLBLK IO\$_WRITELBLK	IO\$_READVBLK IO\$_WRITEVBLK

The set mode I/O function has a symbolic value of IO\$\_SETMODE.

Function codes are defined for all supported devices. Although some of the function codes (for example, IO\$\_READVBLK and IO\$\_WRITEVBLK) are used with several types of devices, most are device-dependent. That is, they perform functions specific to particular types of devices. For example, IO\$\_CREATE is a device-dependent function code; it is used only with file-structured devices such as disks and magnetic tapes. Chapters 2 through 8 provide complete descriptions of the functions and function codes.

#### 1.7.2 Function Modifiers

The high-order 10 bits of the function value are function modifiers. These are individual bits that alter the basic operation to be performed. For example, the function modifier IO\$\_M\_NOECHO can be specified with the function IO\$\_READLBLK to a terminal. When used

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

together, the two values are written as `IO$ READLBLK!IO$M NOECHO`. This means that data typed at the terminal keyboard is entered in the user buffer but not echoed to the terminal. Figure 1-5 shows the format of function modifiers.

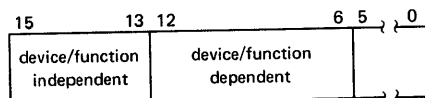


Figure 1-5 Function Modifier Format

As shown, bits 13 through 15 are device/function independent bits, and bits 6 through 12 are device/function dependent bits. Device/function dependent bits have the same meaning, whenever possible, for different device classes. For example, the function modifier `IO$M ACCESS` is used with both disk and magnetic tape devices to cause a file to be accessed during a create operation. Device/function dependent bits always have the same function within the same device class.

There are two device/function independent modifier bits: `IO$M_INHRETRY` and `IO$M_DATACHECK` (a third bit is reserved). `IO$M_INHRETRY` is used to inhibit all error recovery. If any error occurs, and this modifier bit is specified, the operation is immediately terminated and a failure status is returned in the I/O status block (see Section 1.9.2). `IO$M_DATACHECK` is used to compare the data in memory with that on a disk or magnetic tape.

### 1.8 ISSUING I/O REQUESTS

This section describes the entire process involved in issuing I/O requests, including: assigning channels, allocating devices, and issuing QIO requests; the `$QIO`, `$QIOW`, `$INPUT`, and `$OUTPUT` macros; and, finally, status returns.

#### 1.8.1 Channel Assignments

Before I/O requests can be made to a device, the user must assign a channel to establish a link between the user process and the device. A channel is a communication path associated with a device during VAX/VMS I/O operations. The process uses the channel to transfer information to and from the device.

The Assign I/O Channel (`$ASSIGN`) system service is used to assign a channel to a device. To code a call to the `$ASSIGN` system service, the user must supply the name of the device (physical device name or logical name) and the address of a word to receive the assigned channel number. The `$ASSIGN` system service returns the channel number. The process can then request an I/O operation by calling the Queue I/O (`$QIO`) system service and specifying, as one of the arguments, the channel number returned by the `$ASSIGN` system service.

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

In the following example, an I/O channel is assigned to the device TTb4. The channel number is returned in the word at TTCHAN.

```
TTNAME: .LONG 20$-10$           ;TERMINAL NAME DESCRIPTOR
        .LONG 10$
10$:    .ASCII /_TTb4/
20$:
TTCHAN: .BLKW 1                 ;TERMINAL CHANNEL NUMBER
        .
        .
        .
$ASSIGN_S DEVNAM=TTNAME,CHAN=TTCHAN
```

If the first character in the device name (devnam) string is an underline character (  ), the name is considered to be a physical device name; otherwise, one level of logical name translation is performed and the equivalence name, if any, is used.

The Create Mailbox and Assign Channel (\$CREMBX) system service provides another way to assign a channel to a device. In this case, the device is a mailbox. \$CREMBX creates a mailbox and then assigns a channel to it (see Section 7.2.2).

The QIO system service can be performed only on assigned I/O channels and only from access modes that are equal to or more privileged than the access mode from which the original channel assignment was made.

### 1.8.2 Device Allocation

A device can be allocated to a process (or subprocess) by the Allocate Device (\$ALLOC) system service. The allocated device is reserved for the exclusive use of the requesting process, any subprocesses it creates, and subprocesses created by any related subprocess. No other process can allocate the device until the owning process explicitly deallocates it.

Channels can be assigned to both allocated and nonallocated devices; however, a process cannot assign a channel to a device that is allocated to another process. When a channel is assigned to a nonallocated, nonshareable device (for example, a line printer or a magnetic tape device) VAX/VMS implicitly allocates the device.

Access to device functions is controlled by physical and logical I/O privileges, the volume protection mask, and the mountability of the device (a device is mountable if a MOUNT command can be issued for it). Even though a device is allocated to a process, the process cannot perform I/O operations on the device unless access is allowed.

### 1.8.3 I/O Function Requests

After a channel has been assigned, the process can request I/O functions by using the Queue I/O (\$QIO) system service. The \$QIO system service initiates an input or output operation by queuing a request to a specific device that is assigned to a channel.

Certain requirements must be met before a request is queued. For example, a valid channel number must be included in the request, the request must not exceed relevant quotas, and sufficient dynamic memory must be available to complete the operation. Failure to meet such requirements is indicated by a status return (described below in Section 1.8.8).

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

The number of pending I/O requests, the amount of buffer space, and the number of outstanding ASTs that a process can have are controlled by quotas.

Each I/O request causes an I/O request packet to be allocated from system dynamic memory. Additional memory is allocated under the following circumstances:

- The I/O request function is an ACP function
- The target device is a buffered I/O device
- The target device is a network I/O device

After an I/O request is queued, the system does not require the issuing process to wait for the I/O operation to complete. If the process that issued the QIO request cannot proceed until the I/O completes, an event flag can be used to synchronize I/O completion (see Sections 1.8.6.1 and 1.9.1). In this case, the process should request the Wait for Single Event Flag (\$WAITFR) system service at the point where synchronization must occur: that is, where I/O completion is required.

\$WAITFR specifies an event flag for which the process is to wait. (The \$WAITFR event flag must have the same number as the event flag used in the QIO request.) The process then waits while the I/O operation is performed. On I/O completion, the event flag is set and the process is allowed to resume operation.

Other ways to achieve this synchronization include the use of the \$QIOW system service and ASTs, described in Sections 1.8.5 and 1.9.3, respectively. In addition, the I/O status block can be specified and checked if the user wants to determine whether the I/O operation completed without an error, regardless of whether or not the process waits for I/O completion (see Section 1.9.2.)

The QIO system service is accompanied by up to six device/function independent and six device/function dependent arguments. Section 1.8.6 below describes device/function independent arguments. The device/function dependent arguments (P1 through P6) are potentially different for each device/function combination. However, similar functions that are performed by all devices have identical arguments. Furthermore, all functions performed by a particular class of device are identical. Device/function dependent arguments are described in more detail for the individual devices in Chapters 2 through 8.

### 1.8.4 \$QIO Macro Format

The general format for the \$QIO macro, using position-dependent arguments, is:

```
$QIO_S [efn],chan,func,[iosb],[astadr],[astprm],-  
[p1],[p2],[p3],[p4],[p5],[p6]
```

The first six arguments are device/function independent. If keyword arguments are used, they can be written in any order. Arguments P1 through P6 are device/function dependent. The chan and func arguments must be specified in each request; arguments enclosed in brackets ([ ]) are optional.

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

The following example illustrates a typical QIO request using keyword arguments:

```

$QIO_S      EFN=#1,-                ;EVENT FLAG 1
            CHAN=TTCHAN1,-         ;CHANNEL
            FUNC=#IO$ WRITEVBLK,-  ;VIRTUAL WRITE
            P1=BUFADD,-            ;BUFFER ADDRESS
            P2=#BUFSIZE            ;BUFFER SIZE
    
```

### 1.8.5 \$QIOW Macro Format

The Queue I/O Request and Wait For Event Flag (\$QIOW) system service macro combines the \$QIO and \$WAITFR system services. It eliminates any need for explicit I/O synchronization by automatically waiting until the I/O operation is completed before returning control to the process. Thus, \$QIOW provides a simpler way to synchronize the return to the originating process when the process cannot proceed until the I/O operation is completed.

The \$QIOW macro has the same device/function independent and device/function dependent arguments as the \$QIO macro:

```

$QIOW_S [efn],chan,func,[iosb],[astadr],[astprm],-
        [p1],[p2],[p3],[p4],[p5],[p6]
    
```

### 1.8.6 \$QIO and \$QIOW Arguments

Table 1-2 lists the \$QIO and \$QIOW device/function independent arguments and their meanings. Additional information is provided in the paragraphs following the table and in the VAX/VMS System Services Reference Manual.

Table 1-2  
Device/Function Independent Arguments

Argument	Meaning
efn (event flag number)	The number of the event flag that is to be cleared when the I/O function is queued and set when it is completed. This argument is optional in the macro form; if not specified, efn defaults to 0.
chan (channel number)	The number of the I/O channel to which the request is directed. The channel number is obtained from either the \$ASSIGN or \$CREMBX system service. This argument is mandatory in the macro form.
func (function value)	The 16-bit function code and modifier value that specifies the operation to be performed. This argument is mandatory in the macro form.
iosb (I/O status block)	The address of a quadword I/O status block to receive the final I/O status. This argument is optional in the macro form.

(Continued on next page)

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

Table 1-2 (Cont.)  
Device/Function Independent Arguments

Argument	Meaning
astadr (AST address)	The entry point address of an AST routine to be asynchronously executed when the I/O completes. This argument is optional in the macro form.
astprm (AST parameter)	The 32-bit value to be passed to the AST routine as an argument when the I/O completes. It can be used to assist the routine in identifying the particular AST. This argument is optional in the macro form.

**1.8.6.1 Event Flag Number Argument** - The event flag number (efn) argument is the number of the event flag to be associated with the I/O operation. It is optional in a \$QIO or \$QIOW macro. The specified event flag is cleared when the request is issued and set when the I/O operation completes. The specified event flag is also set if the service terminates without queuing the I/O request.

If the process requested the \$QIOW system service, execution is automatically suspended until the I/O completes. If the process requested the QIO system service (with no subsequent \$WAITFR, \$WFLOR, or \$WFLAND macro), process execution proceeds in parallel with the I/O. As the process continues to execute, it can test the event flag at any point by using the Read Event Flags (\$READEF) system service.

Event flag numbers must be in the range of 0 through 127 (however, event flags 24 through 31 are reserved for system use). If no specific event flag is desired, the efn argument can be omitted from the macro. In that case, efn defaults to 0.

**1.8.6.2 Channel Number Argument** - The channel number (chan) argument represents the channel number of the physical device to be accessed by the I/O request. It is required for all \$QIO and \$QIOW requests. The association between the physical device and the channel is specific to the process issuing the I/O request. The channel number is obtained from the \$ASSIGN or \$CREMBX system service (as described above in Section 1.8.1).

**1.8.6.3 Function Argument** - The function (func) argument defines the logical, virtual, or physical I/O operation to be performed when the \$QIO or \$QIOW system service is requested. It is required for all QIO and QIOW requests. The argument consists of a 16-bit function code and function modifier. Up to 64 function codes can be defined. Function codes are defined for all supported device types; most of the codes are device dependent. The function arguments for each I/O driver are described in more detail in Chapters 2 through 8.

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

**1.8.6.4 I/O Status Block Argument** - The I/O status block (iosb) argument specifies the address of the I/O status block to be associated with the I/O request. It is optional in the QIO and QIOW macros. If omitted, the iosb value is 0 which indicates no iosb address is supplied. This block is a quadword that receives the final completion status of the I/O request. Section 1.9.2 describes the I/O status block in more detail.

**1.8.6.5 AST Address Argument** - The AST address (astadr) argument specifies the entry point address of an AST routine to be executed when the I/O operation is complete. If omitted, the astadr value is 0 which indicates no astadr address is supplied. This argument is optional and can be used to interrupt a process to execute special code at I/O completion. When the I/O operation completes, the AST service routine is CALLED at the address specified in the astadr argument. The AST service routine is then executed in the access mode from which the QIO service was requested.

**1.8.6.6 AST Parameter Argument** - The AST parameter (astprm) argument is an optional, 32-bit arbitrary value that is passed to the AST service routine when I/O completes, to assist the routine in identifying the particular AST. A typical use of the astprm argument might be the address of a user control block. If omitted, the astprm value is 0.

**1.8.6.7 Device/Function Dependent Arguments** - Up to six device/function dependent arguments (P1 through P6) can be included in each QIO request. The arguments for terminal read function codes show a typical use of P1 through P6:

- P1 = buffer address
- P2 = buffer size
- P3 = timeout count (for read with timeout)
- P4 = read terminator descriptor block address
- P5 = prompt string buffer address
- P6 = prompt string buffer size

P1 is always treated as an address. Therefore, in the S form of the macro, P1 always generates a PUSHAB instruction. P2 through P6 are always treated as values. In the S form of the macro, these arguments always generate PUSHL instructions.

Inclusion of the device/function dependent arguments in a QIO request depends on the physical device unit and the function specified. A user who wants to specify only a channel, an I/O function code, and an address for AST routine might issue the following:

```
$QIO_S  CHAN=XYCHAN, FUNC=#IO$ READVBLK,-  
        ASTADR=XYAST, P1=BUFADR, P2=#BUFLEN
```

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

In this example, XYCHAN is the address of the word containing the channel to which the request is directed; IO\$ READVBLK is the function code; and XYAST is the AST entry point address. BUFADR and BUFLen are the device/function dependent arguments for an input buffer.

### 1.8.7 \$INPUT and \$OUTPUT Macro Format and Arguments

The \$INPUT and \$OUTPUT macros simplify the use of the \$QIOW macro. These macros generate code to perform virtual operations, using the IO\$ READVBLK and IO\$ WRITEVBLK function codes (the function code is automatically specified in the request), and wait for I/O completion. The macro formats and arguments are:

```
$INPUT  chan,length,buffer,[iosb],[efn]
$OUTPUT chan,length,buffer,[iosb],[efn]
```

Table 1-3 lists the \$INPUT and \$OUTPUT arguments and their meanings.

Table 1-3  
\$INPUT and \$OUTPUT Arguments

Argument	Meaning
chan	The channel on which the I/O operation is to be performed.
length	The length of the input or output buffer.
buffer	The address of the input or output buffer.
iosb	The address of the quadword that receives the completion status of the I/O operation. This argument is optional.
efn	The number of the event flag for which the process waits. This argument is optional; if not specified, efn defaults to 0.

Both the iosb and efn arguments are optional; all other arguments must be included in each macro. Note that the order of the len and buffer arguments is opposite that of the QIO and QIOW P1 and P2 arguments. Also note that \$INPUT and \$OUTPUT do not have the astadr and astprm arguments; neither of these operations can conclude in an AST.

### 1.8.8 Status Returns for System Services

On completion of a system service call, the completion status is returned as a longword value in register R0, shown in Figure 1-6. (System services save the data in all registers except R0 and R1.)

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

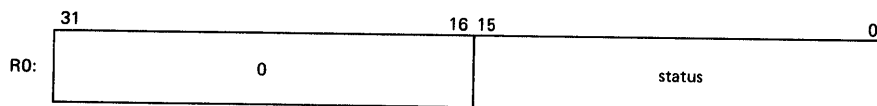


Figure 1-6 System Service Status Return

Completion status is indicated by a value in bits 0 through 15. The low-order 3 bits are encoded with the error severity level; all successful returns have an odd value:

- 0 = warning
- 1 = success
- 2 = error
- 3 = informational (nonstandard) success
- 4 = severe error
- 5-7 = reserved

Each numeric status code has a symbolic name in the form `SS$code`. For example, the return might be `SS$NORMAL`, which indicates successful completion of the system service. There are several error conditions that can be returned. For example, `SS$IVCHAN` indicates that an invalid channel number was specified in an `I/O` request.

The VAX/VMS System Service Reference Manual describes the possible returns for each system service. Table 1-4 lists the valid status returns for the `$QIO`, `$QIOW`, `$INPUT`, and `$OUTPUT` system service requests.

Table 1-4  
\$QIO, \$QIOW, \$INPUT, and \$OUTPUT System Services Status Returns

Status	Meaning
<code>SS\$NORMAL</code>	The <code>\$QIO</code> , <code>\$QIOW</code> , <code>\$INPUT</code> , or <code>\$OUTPUT</code> request was successfully completed; that is, an I/O request was placed in the appropriate device queue.
<code>SS\$ACCVIO</code>	The <code>IOSB</code> , the specified buffer, or the argument list cannot be accessed by the caller.
<code>SS\$EXQUOTA</code>	The buffer quota, buffered I/O quota, or direct I/O quota was exceeded and the process has disabled resource wait mode with the <code>\$SETRWM</code> system service. (The <code>\$SETRWM</code> system service is described in Section 1.4.) <code>SS\$EXQUOTA</code> is also set if the <code>AST</code> quota was exceeded.
<code>SS\$ILLEFC</code>	An illegal event flag number was specified.
<code>SS\$INSMEM</code>	Insufficient dynamic memory is available to complete the service and the process has disabled resource wait mode with the <code>\$SETRWM</code> system service. (The <code>\$SETRWM</code> system service is described in Section 1.4.)

(Continued on next page) .

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

Table 1-4 (Cont.)  
\$QIO, \$QIOW, \$INPUT, and \$OUTPUT System Services Status Returns

Status	Meaning
SS\$_IVCHAN	An invalid channel number was specified; that is, a channel number larger than the number of channels available.
SS\$_NOPRIV	The specified channel was assigned from a more privileged access mode, the channel is not assigned, or the user does not have the proper privilege to access the device.
SS\$_UNASEFC	A common event flag in an unassociated event flag cluster was specified.

Status returns for systems services are not the same as the I/O status returns described in Chapters 2 through 8 for the different I/O drivers (see Section 1.9). A system service status return is the status of the \$QIO, \$QIOW, \$INPUT, \$OUTPUT, or other system service call after completion of the service, that is, after the system returns control to the user. A system service status return does not reflect the completion (successful or unsuccessful) of the requested I/O operation. For example, a \$QIO system service read request to a terminal might be successful (status return is SS\$\_NORMAL) but fail because of a device parity error (I/O status return is SS\$\_PARITY). System service error status return codes refer only to failures to invoke the service.

An I/O status return is the status at the completion of the I/O operation. It is returned in the quadword I/O status block (IOSB). Although some of the symbolic names (for example, SS\$\_NORMAL and SS\$\_ACCVIO) can be used in both types of status returns, they have different meanings.

### 1.9 I/O COMPLETION

Whether an I/O request completed successfully or unsuccessfully can be denoted by one or more return conditions. The selection of the return conditions depends on the arguments included in the QIO macro call. The three primary returns are:

- Event flag--an event flag is set on completion of an I/O operation.
- I/O status block--if the iosb argument was specified in the QIO macro call, a code identifying the type of success or failure is returned in bits 0 through 15 of a quadword I/O status block on completion of the I/O operation. The location of this block is indicated by the user-supplied iosb argument.
- Asynchronous system trap--if an AST address argument was specified in the I/O request, a call to the AST service routine occurs, at the address indicated, on completion of the I/O operation. (The I/O status block, if specified in the I/O request, is updated prior to the AST call.)

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

### 1.9.1 Event Flags

Event flags are status posting bits used by the \$QIO, \$QIOW, \$INPUT, and \$OUTPUT system services to indicate the completion or occurrence of an event. The system service clears the event flag when the operation is queued and sets it when the operation is completed. Event flag services allow users to set or clear certain flags, test the current status of flags, or place a program in a wait state pending the setting of a flag or group of flags.

See the VAX/VMS System Services Reference Manual for more information on event flags and their use.

### 1.9.2 I/O Status Block

The completion status of an I/O request is returned in the first word of the I/O status block (IOSB), as shown in Figure 1-7.

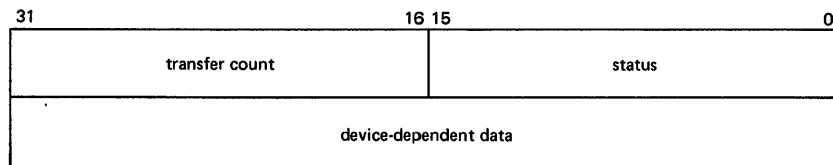


Figure 1-7 I/O Status Block Format

The IOSB indicates whether the operation was successfully completed, the amount of data transferred, and additional device-dependent information such as the number of lines printed. The status return code has the same format and bit significance (bit 0 set indicates success; bit 0 clear indicates error) as the system service status code (see Section 1.8.8). For example, if the process attempts to access a nonexistent disk, a status code of SS\$\_NONE\_XDRV is returned in the I/O status block. The status returns for the individual I/O drivers are listed in Chapters 2 through 8.

The upper half of the first IOSB longword contains the transfer count on completion of the I/O operation if the operation involved the transfer of data to or from a user buffer. For example, if a read operation is performed on a terminal, the number of bytes typed before a carriage return is indicated here. If a magnetic tape unit is the device and a read function is specified, the transfer count represents the number of bytes actually transferred. The second longword of the IOSB can contain certain device-dependent information. This information is supplied in more detail for each I/O driver in Chapters 2 through 8.

The status can be tested symbolically, by name. For example, the SS\$\_NORMAL status is returned if the operation was completed successfully. The following example illustrates the examination of the I/O status block XYIOSB to determine if an error occurred:

```
$QIO_S   CHAN=XYCHAN, FUNC=#IO$_WRITEVBLK,-
          IOSB=XYIOSB, P1=BUFADR, P2=#BUFLN
BLBC     R0,REQERR                      ;CHECK SYSTEM SERVICE
                                              ;STATUS CODE
.
.
.
CMPW     #SS$_NORMAL,XYIOSB             ;CHECK I/O STATUS
                                              ;CODE
BNEQ     ERROR
```

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

The status block can be omitted from a QIO request if the user wishes to assume successful completion of the request and does not want to know how many bytes were transferred. If specified, the IOSB is cleared when the QIO request is issued and then filled with the final status at I/O completion.

### 1.9.3 Asynchronous System Traps

As an option, an AST routine can be specified in the QIO request if the user wants to interrupt the normal execution of a process to execute special code on completion of the request. Even if the process is blocked for a \$WAITFR or \$QIOW, it will be interrupted. When the I/O operation completes, a CALL instruction is used to transfer control to the AST service routine at the entry point address specified in the QIO astadr argument. The address must be the address of an entry mask. The AST service routine is then executed at the access mode from which the QIO request was issued. Figure 1-8 shows the argument list for the CALL instruction.

5
astprm
R0
R1
PC
PSL

Figure 1-8 CALL Instruction Argument List

Using an AST to signal I/O completion allows the process to be occupied with other functions during the I/O operation. The process need not wait until some event occurs before proceeding to another operation.

See the VAX/VMS System Services Reference Manual for more detailed information on ASTs and their use.

### 1.10 DEVICE INFORMATION

Two system services can be used to obtain information about devices: Get Channel Information (\$GETCHN) and Get Device Information (\$GETDEV) system services. The information obtained includes such categories as device characteristics, device type, error count, and operation count.

The Get Channel Information (\$GETCHN) system service is used to obtain information about a device to which an I/O channel is assigned. The \$GETCHN system service can be performed only on assigned channels and only from access modes that are equal to, or more privileged than, the access mode from which the original channel assignment was made.

The Get Device Information (\$GETDEV) system service is used to obtain information about any device.

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

\$GETCHN and \$GETDEV return both primary and secondary device characteristics. Usually, these characteristics are identical. However, they can differ in three instances:

1. If the device is a spooled device, the primary characteristics are those of the intermediate device and the secondary characteristics are those of the spooled device. See the VAX/VMS System Manager's Guide for information on spooling.
2. If the device represents a logical link on a network, the secondary characteristics contain information about the link.
3. If the device has an associated mailbox, the primary characteristics are those of the device and the secondary characteristics are those of the mailbox.

The macro format for a \$GETCHN request is:

```
$GETCHN chan,[prilen],[pribuf],[scdlen],[scdbuf]
```

The macro format for a \$GETDEV request is:

```
$GETDEV devnam,[prilen],[pribuf],[scdlen],[scdbuf]
```

Table 1-5 lists the \$GETCHN and \$GETDEV arguments and their meanings.

Table 1-5  
\$GETCHN and \$GETDEV Arguments

Argument	Meaning
chan	The number of the I/O channel to which a \$GETCHN request is directed (this is not an argument for \$GETDEV).
devnam	The address of a string descriptor for the name of the device to which \$GETDEV is directed (this is not an argument for \$GETCHN).
prilen	The address of the word to receive the length of the primary characteristics. This argument is optional.
pribuf	The address of the buffer descriptor for the buffer that is to receive the primary device characteristics. An address of 0 indicates that no buffer is specified. This argument is optional.
scdlen	The address of the word to receive the length of the secondary characteristics. This argument is optional.
scdbuf	The address of the buffer descriptor for the buffer that is to receive the secondary device characteristics. An address of 0 indicates that no buffer is specified. This argument is optional.

## INTRODUCTION TO VAX/VMS INPUT/OUTPUT

Figure 1-9 shows the format of the device information returned in the primary and secondary buffers.

device characteristics		
buffer size	type	class
device dependent information		
offset to device name	unit number	
owner process PID		
owner process UIC		
error count	volume protection mask	
operation count		
		offset to volume label

Figure 1-9 Buffer Format for \$GETCHN and \$GETDEV System Services

In Figure 1-9, offsets are the displacement from the beginning of the buffer to the specified field. Missing fields are denoted by offsets of 0. Both device name and volume label are stored in the buffer as counted strings. They must be located through the use of their respective offset values. Symbolic offsets for all fields are defined by the \$DIBDEF macro. If both a volume label and a device name are returned, the buffer has a length of 64 bytes.

### 1.10.1 \$GETCHN and \$GETDEV Status Returns

As much information as possible is returned for each of the primary and secondary characteristics. If all the information does not fit in the specified buffers, an appropriate status value is returned. Table 1-6 lists the status return values for the \$GETCHN and \$GETDEV system services.

INTRODUCTION TO VAX/VMS INPUT/OUTPUT

Table 1-6  
\$GETCHN and \$GETDEV Status Returns

Status	Meaning
SS\$_NORMAL	The \$GETCHN or \$GETDEV system service successfully completed.
SS\$_ACCVIO	The caller cannot read a buffer descriptor, write a buffer, or access the argument list.
SS\$_IVCHAN	An invalid channel number was specified in the \$GETCHN request, that is, a channel number larger than the number of channels available; the channel is nonexistent.
SS\$_NOPRIV	The caller does not have the privilege to access the specified channel or the channel is unassigned.
SS\$_BUFFEROVF	The \$GETCHN or \$GETDEV system service successfully completed. The device information returned overflowed the buffer(s) provided and has been truncated.

## CHAPTER 2

### TERMINAL DRIVER

This chapter describes the use of the VAX/VMS terminal driver. This driver supports the DZ-11 Asynchronous Serial Line Multiplexer and the console terminal.

#### 2.1 SUPPORTED TERMINAL DEVICES

Each DZ-11 multiplexer interfaces 8 or 16 asynchronous serial communication lines for use with terminals. It supports programmable baud rates; however, input and output speeds must be the same. VAX/VMS supports the DZ-11 internal modem control.

The system console terminal is attached to the processor with a special purpose interface.

#### 2.2 TERMINAL DRIVER FEATURES AND CAPABILITIES

The VAX/VMS terminal driver provides the following capabilities:

- Type-ahead
- Specifiable or default line terminators
- Special operating modes, such as NOECHO and PASSALL
- American National Standard escape sequence detection
- Terminal/mailbox interaction
- Terminal control characters and special keys
- Dial-up
- Optional parity specification

##### 2.2.1 Type-ahead

Input (data received) from a VAX/VMS terminal is always independent of concurrent output (data sent) to a terminal. This capability is called type-ahead. Type-ahead is allowed on all terminals unless explicitly disabled by the Set Mode characteristic, inhibit type-ahead (TT\$M\_NOTYPEAHD; see Section 2.4.3).

## TERMINAL DRIVER

Data typed at the terminal is retained in the type-ahead buffer until the user program issues an I/O request for a read operation. At that time, the data is transferred to the program buffer and echoed at the terminal where it was typed.

Deferring the echo until a read operation is active allows the user process to specify function code modifiers that modify the read operation. These modifiers can include, for example, `noecho` (`IO$M_NOECHO`) and `convert lowercase characters to uppercase` (`IO$M_CVTLOW`) (see Section 2.4.1.1).

If a read operation is already in progress when the data is typed at the terminal, the data transfer and echo are immediate.

The action of the driver when the type-ahead buffer fills depends on the Set Mode characteristic `TT$M_HOSTSYNC` (see Section 2.4.3). If `TT$M_HOSTSYNC` is not set, `CTRL/G` (BELL) is returned to inform the user that the type-ahead buffer is full. If `TT$M_HOSTSYNC` is set, the driver stops input by sending a `CTRL/S` and the terminal responds by sending no more characters. These warning operations are begun 8 characters before the type-ahead buffer fills. The driver sends a `CTRL/O` to restart transmission when the type-ahead buffer empties completely.

The VAX/VMS System Manager's Guide describes the type-ahead buffer size.

### 2.2.2 Line Terminators

A line terminator is the control sequence that the user types at the terminal to indicate the end of an input line. Optionally, the user process can specify a particular line terminator or class of terminators for read operations.

Terminators are specified by an argument to the QIO request for a read operation. By default, they can be any ASCII control character except `FF`, `VT`, `TAB`, or `BS`. If included in the request, the argument is a user-selected group of characters (see Section 2.4.1.2).

All characters are 7-bit ASCII characters unless data is input on an 8-bit terminal (see Section 2.4.1). (The characteristic `TT$M_EIGHTBIT` determines whether the terminal uses the 7-bit or 8-bit character set; see Table 2-4.) All input characters are tested against the selected terminator(s). The input is terminated when a match occurs or the user's input buffer fills.

### 2.2.3 Special Operating Modes

The VAX/VMS terminal driver supports many special operating modes for terminal lines. Section 2.4.3 lists these modes. All special modes are enabled or disabled by the Set Mode QIO.

### 2.2.4 Escape Sequences

Escape sequences are strings of two or more characters, beginning with the escape character (decimal 27 or hexadecimal 1B), that indicate that control information follows. Many terminals send and respond to such escape sequences to request special character sets or to indicate the position of a cursor.

## TERMINAL DRIVER

The Set Mode characteristic TT\$M ESCAPE (see Section 2.4.3) is used to specify that VAX/VMS terminal lines can generate valid escape sequences. If this characteristic is set, the terminal driver verifies the syntax of the escape sequences. The sequence is always considered a read function terminator and is returned in the read buffer, that is, a read buffer can contain other characters that are not part of an escape sequence, but an escape sequence always comprises the last characters in a buffer. The return information in the read buffer and I/O status block includes the position and size of the terminating escape sequence in the data record (see Section 2.5).

Any escape sequence received from the terminal is checked for correct syntax. If the syntax is not correct, SS\$ BADESCAPE is returned as the status of the I/O. If the escape sequence does not fit in the user buffer, SS\$ PARTESCAPE is returned. The remaining characters are transmitted on the next read. No syntax integrity is guaranteed across read operations. Escape sequences are never echoed. Valid escape sequences are any of the following forms (hexadecimal notation):

`(ESC) <int>...<int> <fin>`

where:

`(ESC)` is pressing the `(ESC)` key, a byte (character) of 1B

`<int>` is an "intermediate character" in the range of 20 to 2F. This range includes the character "space" and 15 punctuation marks. An escape sequence can contain any number of intermediate characters, or none.

`<fin>` is a "final character" in the range of 30 to 7E. This range includes uppercase and lowercase letters, numbers, and 13 punctuation marks.

There are five additional escape sequence forms:

`(ESC) <;> <20-2F>...<30-7E>`  
`(ESC) <?> <20-2F>...<30-7E>`  
`(ESC) <P> <20-2F>...<30-7E>`  
`(ESC) <O> <20-2F>...<40-7E>`  
`(ESC) <Y> <20-7E>...<20-7E>`

For example, when the IDENTIFY escape sequence, escape Z, is sent to a VT-55 terminal, the response from the terminal is `(ESC)<C>`. (Escape sequences are neither displayed nor echoed on the terminal.)

Section 2.2.6 describes control character functions during escape sequences.

### 2.2.5 Terminal/Mailbox Interaction

Mailboxes are virtual I/O devices used for communication between processes. The terminal driver can use a mailbox to communicate with a user process. Chapter 7 describes the mailbox driver.

A user program can use the \$ASSIGN system service to associate a mailbox with one or more terminals. The terminal driver sends messages to this mailbox when terminal-related events occur that require the attention of the user image.



## TERMINAL DRIVER

Interaction with a mailbox associated with a terminal occurs through standard QIO functions and ASTs. Therefore, the process need not have outstanding read requests to an interactive terminal to respond to the arrival of unsolicited data. The process need only respond when the mailbox signals the availability of unsolicited data. Section 2.6 contains an example of mailbox programming.

The ratio of terminals to mailboxes is not always one to one. One user process can have many terminals associated with a single mailbox.

### 2.2.6 Control Characters and Special Keys

A control character is a character that controls action at the terminal rather than passes data to a process. An ASCII control character has a code between 0 and 31, plus 126 and 127 (hexadecimal 0 through 1F, plus 7E and 7F). That is, all normal control characters plus DELETE and ALTMODE. Some control characters are typed at the terminal by simultaneously pressing the CTRL key and a character key, that is, **CTRL/S**. Other control characters, for example, RETURN, LINE FEED, and ESCAPE, are typed by pressing a single key, that is, **RET**, **LF** and **ESC**. Table 2-1 lists the VAX/VMS terminal control characters (none of these characters are interpreted in the PASSALL mode). Table 2-2 lists special terminal keys.

Table 2-1  
Terminal Control Characters

Control Character	Meaning
<b>CTRL/C</b>	<p>Gains the attention of the enabled process if the user program has enabled a CTRL/C AST. If CTRL/C AST is not enabled, <b>CTRL/C</b> is converted to <b>CTRL/Y</b> (see Section 2.4.3).</p> <p>If echo is not disabled, the terminal performs a newline (return followed by a line feed), types ^C, and performs another newline. If <b>CTRL/C</b> is converted to <b>CTRL/Y</b>, then ^Y is echoed.</p> <p>Additional consequences of <b>CTRL/C</b> are:</p> <ul style="list-style-type: none"> <li>• The type-ahead buffer is flushed.</li> <li>• <b>CTRL/S</b> and <b>CTRL/O</b> are reset.</li> <li>• The current I/O operation (if any) is successfully completed. The status return is SS\$_CONTROLC.</li> </ul>
<b>TAB</b> (CTRL/I)	<p>Tabs horizontally. Advances to the next tab stop on terminals with the characteristic TT\$_MECHTAB, but the driver assumes tab stops on MODULO(8), that is, multiples of 8, cursor positions. On terminals without this characteristic, enough spaces are output to move the cursor to the next MODULO (8) position.</p>
<b>LF</b> (CTRL/J)	<p>Performs line feed; filled if TT\$_LFFILL is set.</p>

(continued on next page)

TERMINAL DRIVER

Table 2-1 (Cont.)  
Terminal Control Characters

Control Character	Meaning
<p>(VT) (CTRL/K)</p>	<p>Terminal performs a vertical tab.</p>
<p>(FF) (CTRL/L)</p>	<p>Performs form feed. The driver sends enough LFs (filled) to move the paper to the top of form position described by the length of the page and the current position on the page. The Set Mode function can be used to set page length (see Section 2.4.3). (VAX/VMS does not support terminals with mechanical form feed.)</p>
<p>(CTRL/O)</p>	<p>Discards output. Action is immediate. All output is discarded until the next read operation, the next write operation with a IO\$M_CANCTRLO modifier, or the receipt of the next (CTRL/O). If echo is not disabled, the terminal echoes ^O, followed by a newline. The current write operation (if any), and write operations performed while (CTRL/O) is in effect, are completed with a status return of SS\$_CONTROLO.</p> <p>(CTRL/O), which reenables output, cancels (CTRL/S). (CTRL/C) and (CTRL/Y) cancel (CTRL/O).</p>
<p>(CTRL/O)</p>	<p>Controls data flow; used by terminals and the driver. Restarts data flow to or from a terminal if previously stopped by (CTRL/S). The action occurs immediately with no echo. (CTRL/O) is also used to explicitly solicit read operations.</p> <p>(CTRL/O) is meaningless if the line does not have the characteristic TT\$_TTSYNC, the characteristic TT\$_HOSTSYNC, the characteristic TT\$_READSYNC, or is not currently stopped by (CTRL/S).</p>
<p>(CTRL/R)</p>	<p>Displays current input. When (CTRL/R) is typed during a read operation, a newline is echoed, and the current contents of the input buffer is displayed. If the current operation is a read with prompt (IO\$_READPROMPT) operation, the current prompt string is also displayed. (CTRL/R) has no effect if the characteristic TT\$_NOECHO is set.</p>
<p>(CTRL/S)</p>	<p>Controls data flow; used by both terminals and the driver. (CTRL/S) stops all data flow; the action occurs immediately with no echo. (CTRL/S) is also used to stop read operations. (CTRL/S) is meaningful only if the terminal has the TT\$_TTSYNC, TT\$_HOSTSYNC or TT\$_READSYNC characteristic.</p>

(continued on next page)

## TERMINAL DRIVER

Table 2-1 (Cont.)  
Terminal Control Characters

Control Character	Meaning
CTRL/U	Purges current input data. When CTRL/U is typed before the end of a read operation, the current input is flushed. If echo is not disabled, the terminal echoes ^U, followed by a newline. The prompt string is displayed again if the current operation is a read with prompt (IO\$_READPROMPT).
CTRL/X	Purges the type-ahead buffer and performs a CTRL/U operation. Action is immediate. Inserts a CTRL/U in the data stream if a read operation is in progress.
CTRL/Y	<p>CTRL/Y is a special interrupt or attention character that is used to gain the attention of the command interpreter for a logged-in process. CTRL/Y can be enabled only from supervisor mode by an enable AST I/O function. (CTRL/Y always gains the attention of the command interpreter from a logged-in terminal.) Typing CTRL/Y results in an AST to an enabled process to signify that the user typed CTRL/Y. The terminal performs a newline, types ^Y, and performs another newline if the AST and echo are enabled. CTRL/Y is ignored (and not echoed) if no process is enabled for the AST.</p> <p>Additional consequences of CTRL/Y are:</p> <ul style="list-style-type: none"> <li>• The type-ahead buffer is flushed.</li> <li>• CTRL/S mode is reset.</li> <li>• The current I/O operation (if any) is successfully completed with a 0 transfer count. The status return is S\$\$_CONTROLY.</li> </ul>
CTRL/Z	Echoes ^Z when CTRL/Z is typed as a read terminator. If CTRL/Z is not a read terminator it echoes as itself (hexadecimal 1A). By convention, CTRL/Z constitutes end-of-file.

**TERMINAL DRIVER**

Table 2-2  
Special Terminal Keys

Control Characters	Meaning
<p>ALTMODE</p>	<p>(Decimal 126 or hexadecimal 7E) Converts to escape on terminals that do not have the lowercase characteristic TT\$M_LOWER set.</p>
<p><span style="border: 1px solid black; border-radius: 50%; padding: 2px;">DEL</span> (DELETE)</p>	<p>(Decimal 127 or hexadecimal 7F) Removes last typed character from input stream. <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">DEL</span> is ignored if there are currently no input characters. Hard copy terminals echo the deleted character enclosed in backslashes. for example, if the character z is deleted, \z\ is echoed (the second backslash is echoed after the next character is typed). CRT terminals echo <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">DEL</span> as a backspace followed by a space and another backspace.</p>
<p><span style="border: 1px solid black; border-radius: 50%; padding: 2px;">ESC</span> (ESCAPE)</p>	<p>If escape sequences are recognized (the Set Mode characteristic TT\$M_ESCAPE is set), pressing <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">ESC</span> signals the beginning of an escape sequence. On these terminals <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">ESC</span> is never echoed; however, on terminals that do not recognize escape sequences, <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">ESC</span> is echoed as a dollar sign (\$) if it was used as a read terminator or as hexadecimal 1B if it was not a read terminator.</p>
<p><span style="border: 1px solid black; border-radius: 50%; padding: 2px;">RET</span> (RETURN)</p>	<p>If used during a read (input) operation, <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">RET</span> echoes a newline. All returns are filled on terminals with TT\$M_CRFILL specified.</p>

2.2.6.1 Character Interpretation - All input characters are interpreted according to their value and the characteristics of the terminal.

Figure 2-2 illustrates the character interpretation process.

# TERMINAL DRIVER

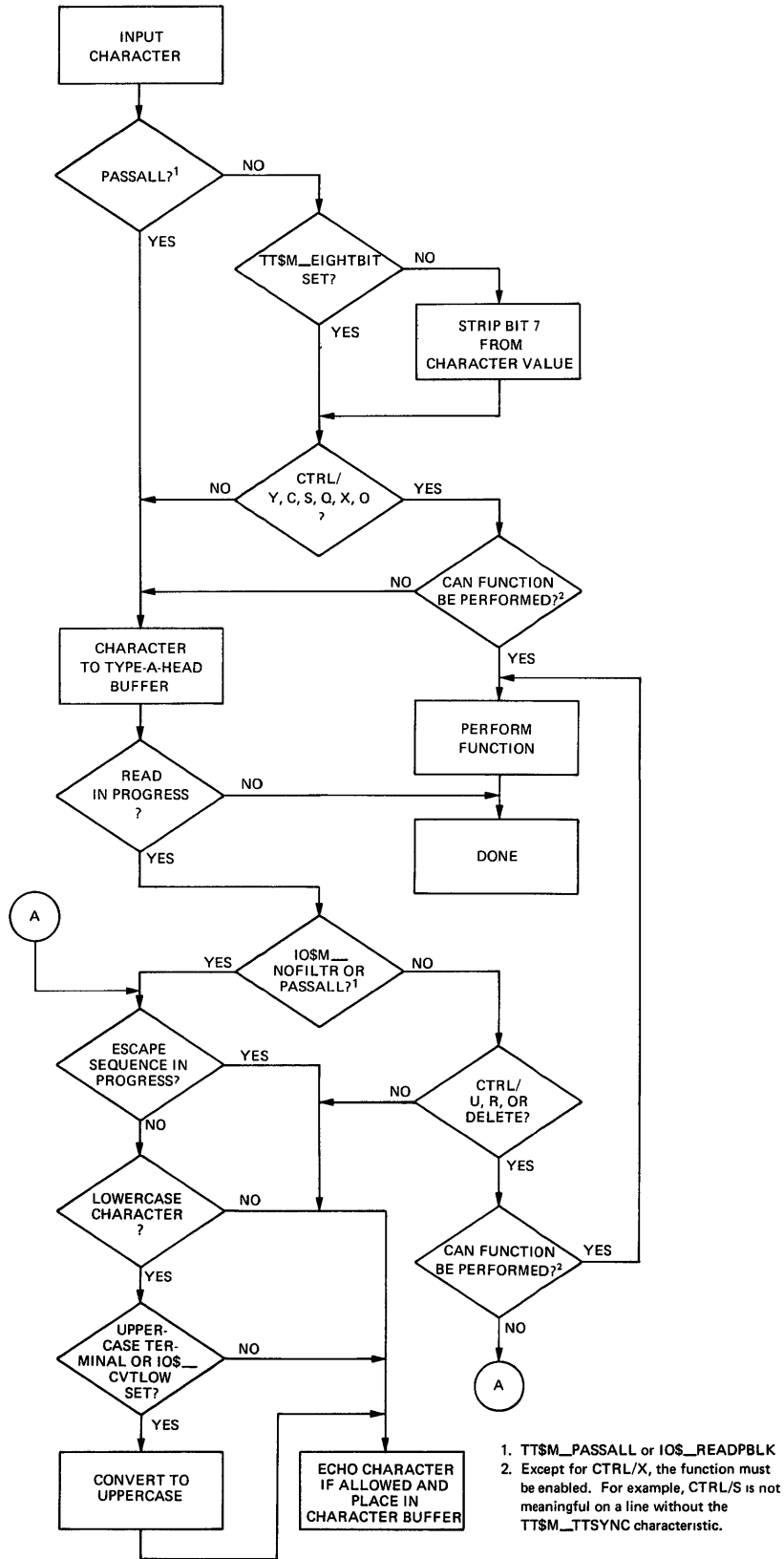


Figure 2-2 Character Interpretation

## TERMINAL DRIVER

### 2.2.7 Dial-up

VAX/VMS supports the DZ-11 internal modem control (for example, Bell 103A, Bell 113, or equivalent) in autoanswer, full-duplex mode. The terminal driver does not support half-duplex operations on modems such as the Bell 202. The terminal characteristic `TT$M_REMOTE` designates the line as being remote to the local computer. The driver automatically sets `TT$M_REMOTE` if the carrier signal changes from off to on.

Dial-up lines are monitored periodically to detect a change in the modem carrier signal. The monitoring period is a system parameter. The VAX/VMS System Manager's Guide describes the dial-up monitoring period.

If a line's carrier signal is lost, the driver waits several monitor periods for the carrier signal to return. If the carrier signal is not detected during this time, the line is "hung-up." The hang-up action signals the owner of the line, through a mailbox message, that the line is no longer in use. (No dial-in message is sent; the unsolicited character message is sufficient when the first available data is received.) The line is not available for two monitor periods after the hang-up sequence begins. The hang-up sequence is not reversible. If the line hangs up, all enabled `CTRL/Y` ASTs are delivered; the `CTRL/Y` AST P2 argument is overwritten with `SS$_HANGUP`. The I/O operation in progress is cancelled and the status value `SS$_ABORT` is returned in the IOSB.

When a line with the `TT$M_REMOTE` characteristic is hung-up, the characteristics of the line are returned to the system default characteristics.

### 2.3 DEVICE INFORMATION

The user process can obtain terminal characteristics by using the `$GETCHN` and `$GETDEV` system services (see Section 1.10). The terminal-specific information is returned in the first three longwords of a user-specified buffer, as shown in Figure 2-3 (Figure 1-8 shows the entire buffer).

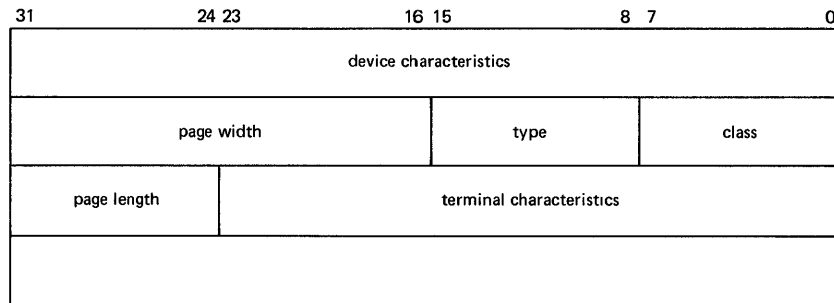


Figure 2-3 Terminal Information

The first longword contains device-independent data. The second and third longwords contain device-dependent data.

Table 2-3 lists the device-independent characteristics returned in the first longword.

## TERMINAL DRIVER

Table 2-3  
Terminal Device-Independent Characteristics

Characteristic Name <sup>1</sup>	Meaning
DEV\$M_AVL	Terminal is on line and available
DEV\$M_IDV	Terminal is capable of input
DEV\$M_ODV	Terminal is capable of output
DEV\$M_SPL	Spooled
DEV\$M_CCL	Carriage control
DEV\$M_REC	Record oriented
DEV\$M_TRM	Terminal device

<sup>1</sup> Defined by the \$DEVDEF macro.

The device class (DC\$TERM) is returned in the first byte of the second longword. The terminal type is returned in the second byte and corresponds to the type of terminal; for example, DT\$VT52. The \$DCDEF macro defines the symbols for terminal class and type. The page width is returned in the third and fourth bytes. The page width can have a value in the range of 0 to 255. A value of 0 has special meaning if the characteristic TT\$M\_WRAP is not set (see Table 2-4).

The third longword contains terminal characteristics and page length. Characteristics are contained in the first three bytes. Page length is contained in the fourth byte. Terminal characteristics are initially set at system generation time to any one of, or any combination of, the values listed in Table 2-4. They can be changed by the Set Mode function (see Section 2.4.3) or by the Set Terminal command. The VAX/VMS Command Language User's Guide describes the Set Terminal command. The \$TTDEF macro defines symbols for terminal characteristics. Page length can have a value in the range of 0 to 255.

## TERMINAL DRIVER

Table 2-4  
Terminal Characteristics

Value <sup>1</sup>	Meaning
TT\$M_CRFILL	Terminal requires fill after <u>RET</u> (the fill type can be specified by the Set Mode function P4 argument).
TT\$M_ESCAPE	Terminal generates escape sequences (see Section 2.2.4). Escape sequences are validated for syntax.
TT\$M_HOLDSCREEN	Terminal is in Holdscreen Mode. The driver automatically causes the terminal to enter or exit from the mode when the mode is changed at the terminal. This mode is meaningful only to DEC VT-52 and VT-55 terminals (see the DECscope User's Manual).
TT\$M_HOSTSYNC	Host/terminal synchronization. <u>CTRL/Q</u> and <u>CTRL/S</u> are used to control data flow and thus keep the type-ahead buffer from filling.
TT\$M_LFFILL	Terminal requires fill after <u>LF</u> (the fill type can be specified by the Set Mode function P4 argument).
TT\$M_LOWER	Terminal has lowercase character set. Unless the terminal is in PASSALL mode, all input, output, and echoed lowercase characters (hexadecimal 61 to 7A) are converted to uppercase if TT\$M_LOWER is not set.
TT\$M_MECHTAB	Terminal has mechanical tabs. In order to accomplish correct line wrapping, MODULO (8) is assumed.
TT\$M_NOECHO	Input characters are not echoed on this terminal line. A physical I/O function is required to change this value. (See Section 2.2.1.)
TT\$M_READSYNC	Read synchronization. All read operations are explicitly solicited by <u>CTRL/Q</u> and terminated by <u>CTRL/S</u> .
TT\$M_TTSYNC	Terminal/host synchronization. Output to the terminal is controlled by terminal-generated <u>CTRL/Q</u> and <u>CTRL/S</u> .
TT\$M_WRAP	A newline should not be inserted if the cursor moves beyond the right margin. If TT\$M_WRAP is not set, no newline is sent.

<sup>1</sup> Prefix can be TT\$M or TT\$V. TT\$M is a mask whose bits correspond to the field set; TT\$V<sub>n</sub> is a bit number.

(Continued on next page)

## TERMINAL DRIVER

Table 2-4 (Cont.)  
Terminal Characteristics

Value <sup>1</sup>	Meaning
TT\$M_PASSALL	Terminal is in PASSALL mode; all input and output data is in binary (no data interpretation occurs). Data termination occurs when the buffer is full or the read data matches the specified terminator. A physical I/O function is required to change this value. (See Section 2.4.1 for a comparison with the read QIO function IO\$_READPBLK.)
TT\$M_REMOTE	Dial-up terminal. Terminal characteristics are returned to the system default when a hang-up occurs on the terminal line. A physical I/O function is required to change this characteristic. The <u>VAX/VMS System Generation Reference Manual</u> describes system default characteristics.
TT\$M_SCOPE	Terminal is a video screen display (CRT terminal).
TT\$M_NOTYPEAHD	Data must be solicited by a read operation. Data is lost if received in the absence of an outstanding read request, that is, unsolicited data. Disables type-ahead capability (see Section 2.2.1).
TT\$M_EIGHTBIT	Terminal uses 8-bit ASCII character set. Terminals without this characteristic use the 7-bit ASCII code. In this case, the eighth bit is masked out on received characters and ignored on output characters. The eighth bit is meaningful only if TT\$M_EIGHTBIT is set.

### 2.4 TERMINAL FUNCTION CODES

The basic terminal I/O functions are read, write, and set mode or characteristics (see Section 1.5). All three I/O functions can take function modifiers. There are two set mode or characteristics functions: Set Mode (IO\$\_SETMODE) and Set Characteristic (IO\$\_SETCHAR).

## TERMINAL DRIVER

### 2.4.1 Read

When a read function code is issued, the user-specified buffer is filled with characters from the associated terminal. VAX/VMS defines four basic read functions, which are listed with their function codes below:

- IO\$\_READVBLK - read virtual block
- IO\$\_READLBLK - read logical block
- IO\$\_READPROMPT - read with prompt
- IO\$\_READPBLK - read physical block (PASSALL)

Read operations are terminated if either of the following conditions occurs:

- The user buffer is full
- The received character is included in a specified terminator class (see Section 2.4.1.2)

The read function codes can take all six device/function-dependent arguments (P1 through P6) on QIO requests:

- P1 = the starting virtual address of the buffer that is to receive the data read
- P2 = the size of the buffer that is to receive the data read in bytes
- P3 = read with timeout, timeout count (see Table 2-5, IO\$\_M\_TIMED)
- P4 = read terminator descriptor block address (see Section 2.4.1.2)
- P5 = the starting virtual address of the prompt buffer that is to be written to the terminal. For read with prompt operations (IO\$\_READPROMPT).
- P6 = the size of the prompt buffer that is to be written to the terminal. For read with prompt operations (IO\$\_READPROMPT).

In a read with prompt operation, the P5 and P6 arguments specify the address and size of a prompt string buffer containing data to be written to the terminal before the input data is read. In a read with prompt operation, a write operation and a read operation are performed on the specified terminal. The prompt string buffer is formatted like any other write buffer, but no carriage control can be implicitly specified. (Carriage control specifiers are described in Section 2.4.2.2.)

During a read with prompt operation, typing **CTRL/D** (which is turned off at the start of any read) stops the prompt string. If **CTRL/U** or **CTRL/X** is typed, the entire prompt string is written out again and the current input is purged. If **CTRL/R** is typed, the current prompt string and input are written to the terminal.

Depending on the terminal type, and the user's input, the prompt string can be very simple or quite complex - from single command prompts to screen fills followed by data input.

## TERMINAL DRIVER

In a read physical block operation, the data received from the associated terminal is placed in the user buffer as binary information without interpretation; the terminal line is in a temporary PASSALL mode. Since IO\$READPBLK is a physical I/O function, it can be specified only by a privileged user (see Section 1.6.1). IO\$READPBLK puts the terminal line in a PASSALL mode which is in effect only for the read physical block operation. This is in contrast with the more comprehensive PASSALL mode established by the Set Mode characteristic TT\$M\_PASSALL. All input and output data is in 8-bit binary format when TT\$M\_PASSALL is set (see Section 2.4.3).

Since IO\$READPBLK does not purge the type-ahead buffer (unless requested using the IO\$M\_PURGE function modifier) the characters in the type-ahead buffer may have been subjected to CTRL/Y/C/S/Q/O interpretation (Section 2.2.6.1). (Characters received while the IO\$READPBLK is in progress are not interpreted.)

2.4.1.1 Function Modifier Codes for Read QIO Functions - Seven function modifiers can be specified with IO\$READVBLK, IO\$READLBLK, IO\$READPROMPT, and IO\$READPBLK. Table 2-5 lists these function modifiers. IO\$M\_CVTLOW and IO\$M\_NOFILTR are not meaningful to IO\$READPBLK.

Table 2-5  
Read QIO Function Modifiers

Code	Consequence
IO\$M_NOECHO	Characters are not echoed (that is, displayed) as they are entered at the keyboard. The terminal line can also be set to a "no echo" mode by the Set Mode characteristic TT\$M_NOECHO, which inhibits all read operation echoing.
IO\$M_CVTLOW	Lowercase alphabetic characters (hexadecimal 61 to 7A) are converted to uppercase when transferred into the user buffer or echoed.
IO\$M_NOFILTR	The terminal driver does not interpret <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">CTRL/U</span> , <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">CTRL/R</span> , or <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">DEL</span> . They are passed to the user.
IO\$M_TIMED	The P3 argument specifies the maximum time (seconds) that can elapse between characters received; that is, the timeout value for the read operation. A value of 0 terminates the read operation, that is, an I/O timeout occurs, if no character is read within 1 second. In effect, data is read from the type-ahead buffer or an error is returned.
IO\$M_PURGE	The type-ahead buffer is purged before the read operation begins.
IO\$M_DSABLMBX	The mailbox is disabled for unsolicited data.
IO\$M_TRMNOECHO	The termination character (if any) is not echoed. There is no formal terminator if the buffer is filled before the terminator is typed.

## TERMINAL DRIVER

**2.4.1.2 Read Function Terminators** - The P4 argument to a read QIO function either specifies the terminator set for the read function or points to the location containing that terminator set. If P4 is 0, all ASCII characters with a code in the range 0 through 31 (hexadecimal 0 through 1F) except LF, VT, FF, TAB, and BS, are terminators. (This is the RMS-32 standard terminator set.)

If P4 does not equal 0, it contains the address of a quadword that either specifies a terminator character bit mask or points to a location containing that bit mask. The quadword has a short form and a long form, as shown in Figure 2-4. In the short form, the correspondence is between the bit number and the binary value of the character; the character is a terminator if the bit is set. For example, if bit 0 is set, NULL is a terminator; if bit 9 is set, TAB is a terminator. If a character is not specified, it is not a terminator. Since ASCII control characters are in the range of 0 through 31, the short form can be used in most cases.

The long form allows use of a more comprehensive set of terminator characters. Any mask size equal to or greater than 1 byte is acceptable. For example, a mask size of 16 bytes allows all 7-bit ASCII characters to be used as terminators; a mask size of 32 bytes allows all 8-bit characters to be used as terminators for 8-bit terminals. An unspecified mask is assumed to be all 0's.

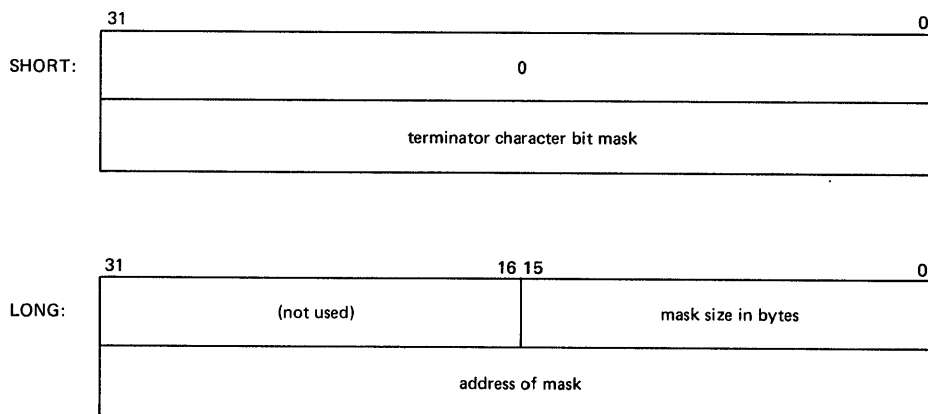


Figure 2-4 Short and Long Forms of Terminator Mask Quadwords

## 2.4.2 Write

Write operations display the contents of a user-specified buffer on the associated terminal. VAX/VMS defines three basic write I/O functions, which are listed with their function codes below:

- IO\$\_WRITEVBLK - write virtual block
- IO\$\_WRITELBLK - write logical block
- IO\$\_WRITEPBLK - write physical block

The write function codes can take the following device/function-dependent arguments:

- P1 = the starting virtual address of the buffer that is to be written to the terminal

## TERMINAL DRIVER

- P2 = the number of bytes that are to be written to the terminal
- P3 (ignored)
- P4 = Carriage control specifier except for write physical block operations. (Write function carriage control is described in Section 2.4.2.2.)

P3, P5, and P6 are not meaningful for terminal write operations.

In write virtual block and write logical block operations, the buffer (P1 and P2) is formatted for the selected terminal and includes the carriage control information specified by P4.

All lowercase characters are converted to uppercase if the characteristics of the selected terminal do not include TT\$M\_LOWER (this does not apply to write physical block operations).

Multiple line feeds are generated for form feeds. The number of line feeds generated depends on the current page position and the length of the page. Multiple spaces are generated for tabs if the characteristics of the selected terminal do not include TT\$M\_MECHTAB (this does not apply to write physical block operations). Tab stops are every 8 characters or positions (that is, 8, 16, 24,...).

**2.4.2.1 Function Modifier Codes for Write QIO Functions - Three function modifiers can be specified with IO\$WRITEVBLK, IO\$WRITELBLK, and IO\$WRITEPBLK. Table 2-6 lists these function modifiers.**

Table 2-6  
Write QIO Function Modifiers

Code	Consequence
IO\$M_CANCTRLO	Turns off <b>CTRL/O</b> (if it is in effect) before the write. Otherwise, the data may not be displayed.
IO\$M_ENABLMBX	Enables use of the mailbox associated with the terminal for notification that unsolicited data is available.
IO\$M_NOFORMAT	Allows nonprivileged users to write information without interpretation or format; in effect the terminal line is in a temporary PASSALL mode.

**2.4.2.2 Write Function Carriage Control - The P4 argument is a longword that specifies carriage control. Carriage control determines the next printing position on the terminal. P4 is ignored in a write physical block operation. Figure 2-5 shows the P4 longword format.**

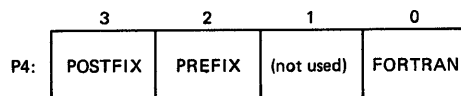


Figure 2-5 P4 Carriage Control Specifier

## TERMINAL DRIVER

Only bytes 0, 2, and 3 in the longword are used. Byte 1 is ignored. If the low-order byte (byte 0) is not 0, the contents of the longword are interpreted as a FORTRAN carriage control specifier. Table 2-7 lists the possible byte 0 values (in hexadecimal) and their meanings.

Table 2-7  
Write Function Carriage Control (FORTRAN: Byte 0 not equal to 0)

Byte Value (hexadecimal)	ASCII Character	Meaning
20	(space)	Single-space carriage control. (Sequence: line feed, print buffer contents, return.)
30	0	Double-space carriage control. (Sequence: line feed, line feed, print buffer contents, return.)
31	1	Page eject carriage control. (Sequence: form feed, print buffer contents, return.)
2B	+	Overprint carriage control. (Sequence: print buffer contents, return.) Allows double printing for emphasis or special effects.
24	\$	Prompt carriage control. (Sequence: line feed, print buffer contents.)
All other values		Same as ASCII space character: single-space carriage control.

If a low-order byte (byte 0) is 0, bytes 2 and 3 of the P4 longword are interpreted as the prefix and postfix carriage control specifiers. The prefix (byte 2) specifies the carriage control before the buffer contents are printed. The postfix (byte 3) specifies the carriage control after the buffer contents are printed. The sequence is:

Prefix carriage control - Print - Postfix carriage control

The prefix and postfix bytes, although interpreted separately, use the same encoding scheme. Table 2-8 shows this encoding scheme in hexadecimal.

**TERMINAL DRIVER**

Table 2-8  
Write Function Carriage Control (P4 byte 0 = 0)

Prefix/Postfix Bytes (Hexadecimal)				Meaning
Bit 7	Bits 0 - 6			
0	0			No carriage control is specified, that is, NULL.
0	1-7F			Bits 0 through 6 are a count of line feeds.
Bit 7	Bit 6	Bit 5	Bits 0-4	Meaning
1	0	0	1-1F	Output the single ASCII control character specified by the configuration of bits 0 through 4 (7-bit character set).
1	1	0	1-1F	Output the single ASCII control character specified by the configuration of bits 0 through 4 which are translated as ASCII characters 128 through 159 (8-bit character set).

Figure 2-6 shows the prefix and postfix hexadecimal coding that produces the carriage control functions listed in Table 2-7. Except for the last example (line skipping), this is an alternative way to achieve these controls.

## TERMINAL DRIVER

	(Space)				
P4:	8D	1	-	0	Sequence: Prefix = NL Print Postfix = CR
	"0"				Sequence: Prefix = LF, LF Print Postfix = CR
P4:	8D	2	-	0	Sequence: Prefix = FF Print Postfix = CR
	"1"				Sequence: Prefix = FF Print Postfix = CR
P4:	8D	8C	-	0	Sequence: Prefix = NULL Print Postfix = CR
	"+"				Sequence: Prefix = NULL Print Postfix = CR
P4:	8D	0	-	0	Sequence: Prefix = LF Print Postfix = NULL
	"\$"				Sequence: Prefix = 24LF Print Postfix = CR
	Example: Skip 24 lines before printing				
P4:	8D	18	-	0	

Figure 2-6 Write Function Carriage Control  
(Prefix and Postfix Coding)

In the first example, the prefix/postfix hexadecimal coding for a single-space carriage control (line feed, print buffer contents, return) is obtained by placing the value 1 in the second (prefix) byte and the sum of the bit 7 value (80) and the return value (D) in the third (postfix) byte:

$$\begin{array}{r}
 80 \text{ (bit 7 = 1)} \\
 + \text{ D (return)} \\
 \hline
 8d \text{ (postfix = return)}
 \end{array}$$

### 2.4.3 Set Mode

Set mode operations affect the operation and characteristics of the associated terminal line. VAX/VMS defines two types of set mode functions:

- Set Mode
- Set Characteristic

## TERMINAL DRIVER

The Set Mode function affects the mode and characteristics of the associated terminal line. Set Mode is a logical I/O function and requires the access privilege necessary to perform logical I/O. A single function code is provided:

- IO\$\_SETMODE

The Set Characteristic function affects the characteristics of the associated terminal line. Set Characteristic is a physical I/O function and requires physical I/O privilege. A single function code is provided:

- IO\$\_SETCHAR

These functions take the following device/function dependent arguments if no function modifiers are specified:

- P1 = address of characteristics buffer
- P2 (ignored)
- P3 = speed specifier
- P4 = fill specifier (bits 0 through 7 CR fill count; bits 8 through 15 LF fill count)
- P5 = parity flags

The P1 argument points to a quadword block, as shown in Figure 2-7. With the exception of terminal characteristics, the contents of the block are the same for both Set Mode and Set Characteristic functions.

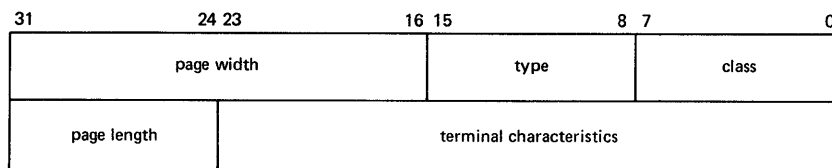


Figure 2-7 Set Mode Characteristics Buffer

The class portion of the block contains DC\$\_TERM, which is defined by the \$DCDEF macro. Type values are defined by the \$DCDEF macro, for example, DT\$\_LA36. Both the page width and page length can have values in the range of 0 to 255. Table 2-9 lists the values for terminal characteristics (Table 2-4 lists their meanings). These values are defined by the \$TTDEF macro.

**TERMINAL DRIVER**

Table 2-9  
Value for Set Mode and Set Characteristic P1 Characteristics

Terminal Characteristics	Applicable to:	
	Set Mode	Set Characteristic
TT\$M_CRFILL	X	X
TT\$M_EIGHTBIT	X	X
TT\$M_ESCAPE	X	X
TT\$M_HOLDSCREEN	X	X
TT\$M_HOSTSYNC	X	X
TT\$M_LFFILL	X	X
TT\$M_LOWER	X	X
TT\$M_MECHTAB	X	X
TT\$M_NOECHO		X
TT\$M_NOTYPEAHD	X	X
TT\$M_PASSALL		X
TT\$M_READSYNC		X
TT\$M_REMOTE		X
TT\$M_SCOPE	X	X
TT\$M_TTSYNC	X	X
TT\$M_WRAP	X	X

The P3 argument specifies the device speed, for example, TT\$C\_BAUD 300. P4 contains fill counts for the carriage return and line feed control characters. Bits 0 through 8 specify the number of fill characters used after a return. Bits 9 through 15 specify the number of fill characters used after a line feed. (P4 is applicable only if TT\$M\_CRFILL or TT\$M\_LFFILL is established as a terminal characteristic; see Table 2-4.)

Three parity flags can be specified in the P5 argument:

- TT\$M\_ALTRPAR - alter parity, change parity on terminal line if set
- TT\$M\_PARITY - enable parity on terminal line if set, disable if clear
- TT\$M\_ODD - parity is odd if set

## TERMINAL DRIVER

If parity is enabled, the interface generates a parity check bit to detect parity mismatch. Parity errors that occur during an I/O read operation are fatal to the operation. Parity errors that occur when no I/O operation is in progress may result in a character loss.

The Set Mode and Set Characteristic functions can take the Enable CTRL/C AST, Enable CTRL/Y AST, and Hang-up function modifiers which are described below.

**2.4.3.1 Hang-up Function Modifier** - The Hang-up function modifier disconnects a terminal that is on a dial-up line. (Dial-up lines are described in Section 2.2.7.) Two combinations of function code and modifier are provided:

- IO\$\_SETMODE!IO\$M\_HANGUP
- IO\$\_SETCHAR!IO\$M\_HANGUP

The Hang-up function modifier takes no arguments.

**2.4.3.2 Enable CTRL/C AST and Enable CTRL/Y AST Function Modifiers** - Both set mode functions can take the Enable CTRL/C AST and Enable CTRL/Y AST function modifiers. These function modifiers request the terminal driver to queue an AST for the requesting process when the user types CTRL/C or CTRL/Y. Four combinations of function code and modifier are provided:

- IO\$\_SETMODE!IO\$M\_CTRLCAST - Enable CTRL/C AST
- IO\$\_SETMODE!IO\$M\_CTRLYAST - Enable CTRL/Y AST
- IO\$\_SETCHAR!IO\$M\_CTRLCAST - Enable CTRL/C AST
- IO\$\_SETCHAR!IO\$M\_CTRLYAST - Enable CTRL/Y AST

These function code modifier pairs take the following device/function-dependent arguments:

- P1 = address of the AST service or 0 if the corresponding AST is to be disabled
- P2 = AST parameter
- P3 = access mode to deliver AST (maximized with caller's access mode)

If the respective enable is in effect, typing CTRL/C or CTRL/Y gains the attention of the enabling process (see Table 2-1). CTRL/Y can be used to gain the attention of the command interpreter and thus allow the user to input special commands such as DUMP, STOP, CONTINUE, etc.

Enable CTRL/C and CTRL/Y AST are single (one-time) enables. After the AST occurs, it must be explicitly re-enabled by one of the four function code combinations described above before the AST can occur again. This function code is also used to disable the AST. The function is subject to AST quotas. A CTRL/Y AST can only be enabled from supervisor mode.

## TERMINAL DRIVER

The user can have more than one CTRL/C or CTRL/Y enabled. All ASTs are given, in their order of request, that is, first in first out, when the character is typed. For example, typing **CTRL/C** results in the delivery of all CTRL/C ASTs.

If no CTRL/C enable is present, the holder of a CTRL/Y enable will receive an AST when **CTRL/C** is typed; line feed, ^Y, return is echoed.

CTRL/C enables are flushed by the Cancel I/O on Channel (\$CANCEL) system service. CTRL/Y enables are flushed only during unit run down, that is, after the last deassignment by the Deassign I/O Channel (\$DASSGN) system service.

Section 2.2.6 describes other effects of **CTRL/Y** and **CTRL/C**.

### 2.5 I/O STATUS BLOCK

The I/O status block formats for the read, write, and set mode I/O functions are shown in Figures 2-8, 2-9, and 2-10, respectively. Table 2-10 lists the possible status returns for these functions.

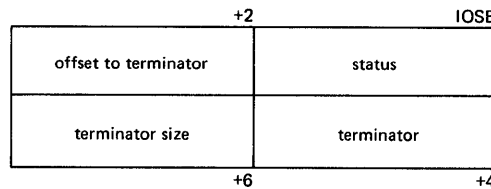
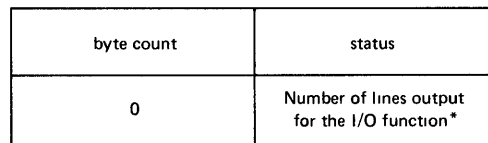


Figure 2-8 IOSB Contents - Read Function

In Figure 2-8, the offset to terminator at IOSB+2 is the count of characters before the terminator character (see Section 2.4.1.2). The terminator character(s) is in the buffer at the offset specified in IOSB+2. When the buffer is full, the offset in IOSB+2 is equal to the requested buffer size. At the same time, IOSB+4 is equal to 0. IOSB+6 contains the size of the terminator string, usually 1. However, in an escape sequence, IOSB+6 contains the size of the validated escape sequence (see Section 2.2.4). The sum of IOSB+2 and IOSB+6 is the number of characters in the buffer.



\*0 if IOS\_WRITEPBLK or PASSALL mode

Figure 2-9 IOSB Contents - Write Function

**TERMINAL DRIVER**

0	speed	status	
0	parity flags	LF fill count	CR fill count

Figure 2-10 IOSB Contents - Set Mode and Set Characteristic Functions

Table 2-10  
Terminal QIO Status Returns

Status	Meaning
SS\$_NORMAL	Successful completion. The operation specified in the QIO was completed successfully. On a read or write operation, the second word of the IOSB can be examined to determine the number of bytes processed. The input or output buffer contains these bytes.
SS\$_TIMEOUT	Operation timeout. The specified terminal could not perform the QIO read operation because a timeout occurred at the terminal, that is, an interrupt was lost, or IO\$M_TIMED was specified on a read operation (see Table 2-5), or a hardware timeout occurred.
SS\$_ABORT	The operation was canceled by the Cancel I/O on Channel (\$CANCEL) system service.
SS\$_PARTEscape	Partial escape sequence was stored. An escape sequence was started but read-buffer space was exhausted before the sequence was completed. The remainder of the sequence is available from the type-ahead buffer on the next read unless the terminal line has the TT\$M_NOTYPEAHD characteristic (see Section 2.2.4).
SS\$_BADEscape	Invalid escape sequence terminator begins at the offset (IOSB+2).
SS\$_CONTROLO	Write operation not completed because <u>CTRL/O</u> was typed.

(Continued on next page)

## TERMINAL DRIVER

Table 2-10 (Cont.)  
Terminal QIO Status Returns

Status	Meaning
SS\$_CONTROL_C	Read and write operation not completed because <b>CTRL/C</b> was typed.
SS\$_CONTROL_Y	Read or write operation not completed because <b>CTRL/Y</b> was typed.
SS\$_PARITY	Parity bit mismatch detected by the device interface during a read operation. The I/O operation stopped when the mismatch was detected. (Data was received up to this point in the operation.) SS\$_PARITY is meaningful only on terminal lines that have parity enabled.

### 2.6 PROGRAMMING EXAMPLE

The following program shows examples of several I/O operations, using mailboxes, direct cursor addressing, and ASTs. The program illustrates some important concepts concerning terminal driver programming: creating a mailbox, assigning a terminal channel, associating the mailbox to the channel, and enabling unsolicited input ASTs and CTRL/C ASTs.

Initially, the program uses direct cursor addressing to write a message on the terminal screen and then hibernates, that is, enters a wait mode. The user then types an entry on the keyboard. The terminal driver sends notice of this unsolicited input to the mailbox associated with the terminal channel and the program is awakened by a mailbox AST. The AST routine reads the mailbox, reads the channel, prints the message, and hibernates again. If the typed character is a **CTRL/C** the CTRL/C AST routine is entered and a different message is printed. Section 7.5 shows another example of mailbox programming.

```

                .TITLE  TERMINAL DRIVER PROGRAMMING EXAMPLE
                .IDENT  /01/
;
; DEFINE NECESSARY SYMBOLS
;
                SIODEF                                ;DEFINE I/O FUNCTION CODES
;
; Allocate storage for necessary data structures
;
; First we have the message that is printed out upon receipt of unsolicited input
;
AST_MESSAGE:
    .BYTE      27,72,27,74                            ;Home, and clear to end of screen
    .BYTE      27,89,42,54                            ;Set X 10 lines down Y to 22 lines across
    .BYTE      0,0,0,0,0,0                            ;Some nulls as fillers
    .ASCII     "UNSOLICITED INPUT RECEIVED"
;
                AST_MSG_SIZE=-AST_MESSAGE              ;Define size of message
;
; Now we have the control C ast message which gets printed every time
; that someone types control C.
;
CTRL_C_MSG:
    .BYTE      10,13                                  ;Line feed carriage return
    .ASCII     "YOU HAVE TYPED CONTROL C" ;Message
;
    .BYTE      10,13                                  ;Line feed carriage return
    CTRL_C_MSG_LEN=-CTRL_C_MSG                      ;Define length of message

```

## TERMINAL DRIVER

```

;
; Now we must allocate some space for the mail
;
BOXBUFF: .BLKW 20 ;Allocate space for mail
         BOXBUFFSIZE=.-BOXBUFF ;Define length of mailbox
;
; Allocate device name string and descriptor
;
DEVICE_DESCR:
         .LONG 20$-10$ ;Length of name string
         .LONG 10$ ;Address of name string
10$: .ASCII "PRINTER_1" ;Name string
20$: ;Reference label
;
; Allocate space to store assigned channel number
;
DEVICE_CHANNEL:
         .BLKW 1 ;
;
; Now allocate space for the terminal input buffer
;
INPUT_BUFFER: .BLKW 1 ;Reply in 1 byte please
              INBUFLNGTH=.-INPUT_BUFFER ;Define buffer size
;
; Now we have the initial prompt message
;
PROMPT_MESSAGE:
         .BYTE 27,72,27,74 ;Home+clear to end of screen
         .ASCII "ARE YOU THERE? " ;Prompt question
         PROMPTSIZE=.-PROMPT_MESSAGE ;Define message size
;
; Now allocate mailbox name string and descriptor
;
BOXNAME: .LONG ENDBOX-NAMEBOX ;Length of mailbox string
         .LONG NAMEBOX ;Address of mbx string
NAMEBOX: .ASCII "146_MAIN_ST" ;Name string
ENDBOX: ;Reference label
;
; Allocate space to store assigned channel number
;
MAILBOX_CHAN: .BLKW 1 ;Channel number goes here
;
; Program starting point. Initially a mailbox is created, then a channel
; with an associated mailbox is assigned to the terminal. A prompt message
; is printed to the terminal and both control C and unsolicited input AST
; recognition is enabled. The program then hibernates until a key is struck
;
START: .WORD 0 ;Entry mask
       $CREMBX_S CHAN=MAILBOX_CHAN,- ;Channel is mailbox
       PROMSK=#^X0000,- ;No protection
       BUFQUO=#^X0060,- ;Buffer quota is hex 60
       MAXMSG=#^X0060,-
       LOGNAM=BOXNAME ;Logical name descriptor
       CMPW #SSS_NORMAL,RO ;Directive OK?
       BSBW ERROR_CHECK ;Find out
       $ASSIGN_S DEVNAM=DEVICE_DESCR,- ;Assign channel
       CHAN=DEVICE_CHANNEL,-
       MBXNAM=BOXNAME ;Associated mailbox descriptor address
       CMPW #SSS_NORMAL,RO ;Directive OK?
       BSBW ERROR_CHECK ;Find out
;
; Specify an ast for wake up and control C delivery
;
       BSBW WAKEUPASTGEN ;Set up unsolicited input recognition
       BSBW ENABLE_CTRLCAST ;And control C recognition
;
; Now do a write of the prompt message
;
       $QIOW_S CHAN=DEVICE_CHANNEL,- ;Channel
       FUNC=#IOS_WRITEVBLK!IOSM_ENABLMBX,- ;Function is write virtual
       P1=PROMPT_MESSAGE,- ;Buffer
       P2=#PROMPTSIZE ;Size of buffer
       CMPW #SSS_NORMAL,RO ;Directive OK?
       BSBW ERROR_CHECK ;Find out
       $HIBER_S ;Hibernate until awoken
;
; This is the unsolicited input AST entry point
;
ASTWAKEUP::
         .WORD 0 ;Entry mask
       $QIOW_S FUNC=#IOS_WRITEVBLK,- ;Function = write physical
         CHAN=DEVICE_CHANNEL,- ;To device_channel
         P1=AST_MESSAGE,- ;P1=buffer containing the AST message
         P2=#AST_MSG_SIZE ;P2=size of message
       CMPW #SSS_NORMAL,RO ;Directive OK?
       BSBW ERROR_CHECK ;Find out

```

## TERMINAL DRIVER

```

;
; Now get the mail
;
      SQIOW_S  FUNC=#IOS_READVBLK!IOSM_NOW,-      ;Read box now!
              CHAN=MAILBOX_CHAN,-              ;Channel=mailbox

              P1=BOXBUFF,-                      ;where to read it
              P2=#BOXBUFFSIZE                  ;How much
      CMPW    #SSS_NORMAL,RO                      ;Directive UK?
      BSBW    ERROR_CHECK                          ;Find out
;
; Normally the mailbox data is decoded at this point to determine
; which terminal sent the wake up AST. This will be omitted
; From this example since there is only one terminal involved.
; The terminal which has the input is now read
;
      SQIOW_S  CHAN=DEVICE_CHANNEL,-            ;Read terminal
              FUNC=#IOS_READVBLK!IOSM_NOECHO,- ;Read terminal with no echo
              P1=INPUT_BUFFER,-              ;where to read it
              P2=#INBUFLNGTH                 ;How much
      CMPW    #SSS_NORMAL,RO                      ;Directive UK?
      BSBW    ERROR_CHECK                          ;Find out
      BSBW    WAKEUPASTGEN                        ;Re-enable AST awareness
      RET                                           ;And return
;
; This is where we come upon receipt of a ^C AST
;
CTRLCAST::
      WORD    0                                  ;Entry mask
      BSBW    ENABLE_CTRLCAST                    ;Re-enable ^C recognition
      SQIOW_S  CHAN=DEVICE_CHANNEL,-            ;Type out message to device_channel
              FUNC=#IOS_WRITEVBLK,-          ;write virtual
              P1=CTRL_C_MSG,-                ;where message is
              P2=#CTRL_C_MSG_LEN            ;Length of it
      CMPW    #SSS_NORMAL,RO                      ;Directive OK?
      BSBW    ERROR_CHECK                          ;Find out
      RET                                           ;Return
;
; This subroutine sets up the mailbox to deliver an AST to the
; User program upon receipt of a message.
; The AST has to be re-enabled after each AST has been processed.
;
WAKEUPASTGEN::
      SQIOW_S  CHAN=MAILBOX_CHAN,-            ;Mailbox channel
              FUNC=#IOS_SETMODE!IOSM_WRTATTN,- ;Set mode function
              P1=ASTWAKEUP,-                 ;Entry point of AST service
              P2=#^XOFFF                    ;AST parameter
      CMPW    #SSS_NORMAL,RO                      ;Directive OK?
      BSBW    ERROR_CHECK                          ;Find out
      RSB                                           ;And return
;
; This is where we enable ^C AST delivery
;
ENABLE_CTRLCAST::
      SQIOW_S  CHAN=DEVICE_CHANNEL,-          ;Channel=device_channel
              FUNC=#IOS_SETMODE!IOSM_CTRLCAST,- ;Function is ^C AST
              P1=CTRLCAST,-                  ;Address of AST routine
              P2=#^XOFFADE,-                 ;AST parameter
              P3=#3                           ;User mode
      CMPW    #SSS_NORMAL,RO                      ;Directive UK?
      BSBW    ERROR_CHECK                          ;Find out
      RSB                                           ;Go back
;
; This is the error checking part of the program. Normally some kind
; Of recovery would be attempted but not for this example.
;
ERROR_CHECK:
      BNEQ    EXIT                                ;Branch if directive failed
      RSB                                           ;Else return
EXIT:      RET                                    ;Return
              .END      START

```

CHAPTER 3  
DISK DRIVERS

This chapter describes the use of the VAX/VMS disk drivers. These drivers support the devices listed in Table 3-1 and detailed in Section 3.1.

Table 3-1  
Disk Devices

Model	Type*	RPM	Surfaces	Cylinders	Bytes/ Track	Drive
RM03	Pack	3600	5	823	16,384	67,420,160
RP05	Pack	3600	19	411	11,264	87,960,576
RP06	Pack	3600	19	815	11,264	174,423,040
RK06	Cart	2400	3	411	11,264	13,888,512
RK07	Cart	2400	3	815	11,264	27,550,480

\*Pack = pack disk; Cart = cartridge disk

### 3.1 SUPPORTED DISK DEVICES

The following sections provide greater detail on each of the disk devices listed in Table 3-1.

#### 3.1.1 RM03 Pack Disk

The RM03 pack disk is a removable, moving head disk that consists of five data surfaces. The RM03 is connected to the system by a MASSBUS adapter (MBA). Up to eight drives can be connected to each MBA.

#### 3.1.2 RP05 and RP06 Pack Disks

The RP05 and RP06 pack disks consist of 19 data surfaces and a moving read/write head. They offer large storage capacity and rapid access time. The RP06 pack disk has nearly twice the capacity of the RP05. These disks are connected to the system by an MBA. Up to eight drives can be connected to each MBA.

## DISK DRIVERS

### 3.1.3 RK06 and RK07 Cartridge Disks

The RK06 cartridge disk is a removable, random-access, bulk-storage device with three data surfaces. The RK07 is a double-density RK06. The RK06 and RK07 are connected to the system by an RK611 controller which interfaces to the UNIBUS adapter (UBA). The subsystem is expandable to eight drives and is suitable for medium- to large-scale systems.

### 3.2 DRIVER FEATURES AND CAPABILITIES

The VAX/VMS disk drivers provide the following capabilities:

- Multiple controllers of the same type; for example, more than one MBA or RK611 can be used on the system
- Up to eight drives per controller
- Different types of drive on a single controller (MBA only)
- Overlapped seeks
- Data checks on a per-request, per-file, and/or per-volume basis
- Full recovery from power failure for on-line drives with volumes mounted
- Extensive error recovery algorithms; for example, error code correction and offset
- Dynamic, as well as static, bad block handling
- Logging of device errors in a file that can be displayed by field service personnel or customer personnel
- On-line diagnostic support for drive level diagnostics
- Multiple block, noncontiguous, virtual I/O operations at the driver level

The following sections describe the data check, overlapped seek, and error recovery capabilities in greater detail.

#### 3.2.1 Data Check

A data check is made after successful completion of a read or write operation, and compares the data in memory with the data on disk to make sure they match.

Disk drivers support data checks at three levels:

- Per request -- users can specify the data check function modifier (IO\$M\_DATACHECK) on a read logical block, write logical block, read virtual block, write virtual block, read physical block, or write physical block I/O operation.

## DISK DRIVERS

- Per volume -- users can specify the characteristics "data check all reads" and/or "data check all writes" when the volume is mounted. The VAX/VMS Command Language User's Guide describes volume mounting and dismounting.
- Per file -- users can specify the file access attributes "data check on read" or "data check on write." File access attributes are specified when the file is accessed. Chapter 9 of this manual and the VAX-11 Record Management Services Reference Manual describe file access.

Offset recovery is performed during a data check but Error Code Correctable (ECC) correction is not (see Section 3.2.3). This means that if a read operation is performed and an ECC correction applied, the data check would fail even though the data in memory is correct. In this case, the driver returns a status code indicating that the operation was successfully completed, but that the data check could not be performed because of an ECC correction.

Data checks on read operations are extremely rare and users can either accept the data as is; treat the ECC correction as an error; or accept the data, but immediately move it to another area on the disk volume.

### 3.2.2 Overlapped Seeks

A seek operation involves moving the disk read/write heads to a specific place on the disk without any transfer of data. All transfer functions, including data checks, are preceded by an implicit seek operation (except when the seek is inhibited by the physical I/O function modifier IO\$M\_INHSEEK). Seek operations can be overlapped. That is, while one drive performs a seek operation, any number of other drives can also perform seek operations.

During the seek operation, the controller is free to perform transfers on other units. Thus, seek operations can also overlap data transfer operations. For example, at any one time, seven seeks and one data transfer could be in progress on a single controller.

This overlapping is possible because, unlike I/O transfers, seek operations do not require the controller once they are initiated. Therefore, seeks are initiated before I/O transfers and other functions that require the controller for extended periods.

### 3.2.3 Error Recovery

Error recovery in VAX/VMS is aimed at performing all possible operations to successfully complete an I/O operation. Error recovery operations fall into four categories:

- Handling special conditions such as power failure and interrupt timeout
- Retrying nonfatal controller and/or drive errors
- Applying error correction information
- Offsetting read heads to try to obtain a stronger recorded signal

## DISK DRIVERS

The error recovery algorithm uses a combination of these four types of error recovery operations to complete an I/O operation.

Power failure recovery consists of waiting for mounted drives to spin up and come on line followed by a re-execution of the I/O operation that was in progress at the time of the power failure.

Device timeout is treated as a nonfatal error. The operation that was in progress when the timeout occurred is re-executed up to eight times before a timeout error is returned.

Nonfatal controller/drive errors are simply re-executed up to eight times before a fatal error is returned.

All normal error recovery (nonspecial conditions) can be inhibited by specifying the inhibit retry function modifier (IO\$M\_INHRETRY). If any error occurs and this modifier is specified, the virtual, logical, or physical I/O operation is immediately terminated, and a failure status is returned. This modifier has no effect on power failure and timeout recovery.

### 3.3 DEVICE INFORMATION

Users can obtain information on all disk device characteristics by using the \$GETCHN and \$GETDEV system services (see Section 1.10). The disk-specific information is returned in the first three longwords of a user-specified buffer, as shown in Figure 3-1 (Figure 1-8 shows the entire buffer).

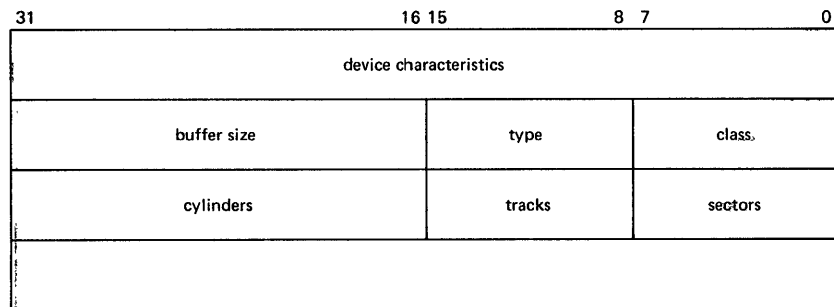


Figure 3-1 Disk Information.

Table 3-2 lists the device characteristics returned in the first longword.

## DISK DRIVERS

Table 3-2  
Disk Device Characteristics

Dynamic Bits <sup>1</sup> (Conditionally Set)	Meaning
DEV\$M_AVL	Device is on line and available
DEV\$M_FOR	Foreign device
DEV\$M_MNT	Volume mounted
DEV\$M_RCK	Perform data check all reads
DEV\$M_WCK	Perform data check all writes
Static Bits <sup>1</sup> (Always Set)	Meaning
DEV\$M_FOD	File-oriented device
DEV\$M_IDV	Device is capable of input
DEV\$M_ODV	Device is capable of output
DEV\$M_RND	Device is capable of random access
DEV\$M_SHR	Device is shareable

<sup>1</sup> Defined by the \$DEVDEF macro.

The second longword contains information on the device class and type, and the buffer size. The device class for disks is DC\$\_DISK and the device types are:

Device Type	Disk
DT\$_RM03	RM03
DT\$_RP05	RP05
DT\$_RP06	RP06
DT\$_RK06	RK06
DT\$_RK07	RK07

THE \$DCDEF macro defines the device type and class names. The buffer size is the default to be used for disk transfers (this default is normally 512 bytes).

The third longword contains information on the number of cylinders per disk, the number of tracks per cylinder, and the number of sectors per track.

## DISK DRIVERS

### 3.4 DISK FUNCTION CODES

VAX/VMS disk drivers can perform logical, virtual, and physical I/O functions.

Logical and physical I/O functions allow access to volume storage and require only that the issuing process have access to the volume. Virtual I/O functions require intervention by an Ancillary Control Process (ACP) and must be executed in a prescribed order. The normal procedure is to create a file and access it. Information is then written to the file and the file is deaccessed. The file is subsequently accessed, the information is read, and the file is deaccessed. The file is deleted when the information it contains is no longer useful.

Any number of blocks (up to a maximum of 65K bytes) can be read or written by a single request. The number itself has no effect on the applicable quotas (direct I/O, buffered I/O, and AST). Reading or writing 1 block or 10 blocks subtracts the same amount from the quota.

The volume to which a logical or virtual function is directed must be mounted in order for the function to actually be executed. If it is not mounted, either a device not mounted or invalid volume status is returned in the I/O status block.

Table 3-3 lists the logical, virtual, and physical disk I/O functions and their function codes. These functions are described in more detail in Appendix A. Chapter 9 describes the QIO level interface to the disk device ACP.

## DISK DRIVERS

Table 3-3  
Disk I/O Functions

Function Code and Arguments	Type <sup>1</sup>	Function Modifiers	Function
IO\$_CREATE P1, [P2], - [P3], [P4], [P5]	V	IO\$M_CREATE IO\$M_ACCESS IO\$M_DELETE	Create a directory entry or a file
IO\$_ACCESS P1, [P2], - [P3], [P4], [P5]	V	IO\$M_CREATE IO\$M_ACCESS	Search a directory for a specified file and access the file if found
IO\$_DEACCESS P1, [P2], - [P3], [P4], [P5]	V		Deaccess a file and, if specified, write final attributes in the file header
IO\$_MODIFY P1, [P2], - [P3], [P4], [P5]	V	.	Modify the file attributes and/or allocation.
IO\$_DELETE P1, [P2], - [P3], [P4], [P5]	V	IO\$M_DELETE	Remove a directory entry and/or file header
IO\$_READVBLK P1, P2, P3	V	IO\$M_DATACHECK IO\$M_INHRETRY	Read virtual block
IO\$_READLBLK P1, P2, P3	L	IO\$M_DATACHECK IO\$M_INHRETRY	Read logical block
IO\$_READPBLK P1, P2, P3	P	IO\$M_DATACHECK IO\$M_INHRETRY IO\$M_INHSEEK	Read physical block
IO\$_WRITEVBLK P1, P2, P3	V	IO\$M_DATACHECK IO\$M_INHRETRY	Write virtual block
IO\$_WRITELBLK P1, P2, P3	L	IO\$M_DATACHECK IO\$M_INHRETRY	Write logical block
IO\$_WRITEPBLK P1, P2, P3	P	IO\$M_DATACHECK IO\$M_INHRETRY IO\$M_INHSEEK	Write physical block
IO\$_SETMODE P1	L		Set disk characteristics for subsequent operations
IO\$_SETCHAR P1	P		Set disk characteristics for subsequent operations

<sup>1</sup> V = virtual; L = logical; P = physical

## DISK DRIVERS

The function-dependent arguments for IO\$\_CREATE, IO\$\_ACCESS, IO\$\_DEACCESS, IO\$\_MODIFY, and IO\$\_DELETE are:

- P1 -- the address of the File Information Block (FIB) descriptor.
- P2 -- the address of the file name string descriptor (optional). If specified, the name is entered in the directory specified by the FIB.
- P3 -- the address of the word that is to receive the length of the resulting file name string (optional).
- P4 -- the address of a descriptor for a buffer that is to receive the resulting file name string (optional).
- P5 -- the address of a list of attribute descriptors (optional). If specified, the indicated attributes are read (IO\$\_ACCESS) or written (IO\$\_CREATE, IO\$\_DEACCESS, and IO\$\_MODIFY).

The function-dependent arguments for IO\$\_READVBLK, IO\$\_READLBLK, IO\$\_WRITEVBLK, and IO\$\_WRITELBLK are:

- P1 -- the starting virtual address of the buffer that is to receive the data in the case of a read operation; or, in the case of a write operation, the virtual address of the buffer that is to be written on the disk.
- P2 -- the number of bytes that are to be read from the disk, or written from memory to the disk. An even number must be specified if the controller is an RK611.
- P3 -- the starting virtual/logical disk address of the data to be transferred in the case of a read operation; or, in the case of a write operation, the disk address of the area that is to receive the data.

In a virtual read or write, the address is expressed as a block number within the file, that is, block 1 of the file is virtual block 1. (Virtual block numbers are converted to logical block numbers via mapping windows set up by the file system ACP process.)

In a logical read or write, the address is expressed as a block number relative to the start of the disk. For example, the first sector on the disk contains (at least the beginning of) block 0.

The function-dependent arguments for IO\$\_READPBLK and IO\$\_WRITEPBLK are:

- P1 -- the starting virtual address of the buffer that is to receive the data in the case of a read operation; or, in the case of a write operation, the starting virtual address of the buffer that is to be written on the disk.
- P2 -- the number of bytes that are to be read from the disk, or written from memory to the disk. An even number must be specified if the controller is an RK611.

## DISK DRIVERS

- P3 -- the starting physical disk address of the data to be read in the case of a read operation; or, in the case of a write operation, the starting physical address of the disk area that is to receive the data. The address is expressed as sector, track, and cylinder in the format shown in Figure 3-2.

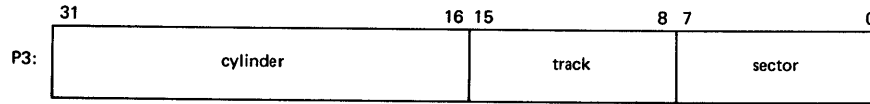


Figure 3-2 Starting Physical Address

The function dependent argument for `IO$_SETMODE` and `IO$_SETCHAR` is:

- P1 -- the address of a quadword device characteristics descriptor

### 3.4.1 Read

This function reads data into a specified buffer from disk starting at a specified disk address.

VAX/VMS provides three read function codes:

- `IO$_READVBLK` - read virtual block
- `IO$_READLBLK` - read logical block
- `IO$_READPBLK` - read physical block

If a read virtual block function is to a volume that is mounted foreign, it is converted to a read logical block function. If a read virtual block function is to a volume that is mounted structured, it is handled in the normal manner for a file-structured device.

Three function-dependent arguments are used with these codes: P1, P2, and P3. These arguments were described above, in the beginning of Section 3.4.

The data check function modifier (`IO$M_DATACHECK`) can be used with all read functions. If this modifier is specified, a data check operation is performed after the read data operation has been completed. A data check operation is also performed if the volume read, or the volume on which the file resides (virtual read), has the characteristic "data check all reads." Furthermore, a data check is performed after a virtual read if the file has the attribute "data check on read."

The inhibit retry function modifier (`IO$M_INHRETRY`) can be used with all read functions. If this modifier is specified, all error recovery attempts are inhibited. `IO$M_INHRETRY` takes precedence over `IO$M_DATACHECK`. If both are specified and an error occurs, there is no attempt at error recovery and no data check operation is performed. If an error does not occur, the data check operation is performed.

## DISK DRIVERS

### 3.4.2 Write

This function writes data from a specified buffer to disk starting at a specified disk address.

VAX/VMS provides three write function codes:

- IO\$\_WRITEVBLK - write virtual block
- IO\$\_WRITELBLK - write logical block
- IO\$\_WRITEPBLK - write physical block

If a write virtual block function is to a volume that is mounted foreign, it is converted to a write logical block function. If a write virtual block function is to a volume that is mounted structured, it is handled in the normal manner for a file-structured device.

Three function-dependent arguments are used with these codes: P1, P2 and P3. These arguments were described above, in the beginning of Section 3.4.

The data check function modifier (IO\$\_M\_DATACHECK) can be used with all write functions. If this modifier is specified, a data check operation is performed after the write data operation has been completed. A data check operation is also performed if the volume written, or the volume on which the file resides (virtual write), has the characteristic "data check all writes." Furthermore, a data check is performed after a virtual write if the file has the attribute "data check on write."

The inhibit retry function modifier (IO\$\_M\_INHRETRY) can be used with all write functions. If this modifier is specified, all error recovery attempts are inhibited. IO\$\_M\_INHRETRY takes precedence over IO\$\_M\_DATACHECK. If both are specified and an error occurs, there is no attempt at error recovery and no data check operation is performed. If an error does not occur, the data check operation is performed.

### 3.4.3 Set Mode

Set mode operations affect the operation and characteristics of the associated disk device. VAX/VMS defines two types of set mode functions:

- Set Mode
- Set Characteristic

**3.4.3.1 Set Mode** - The Set Mode function affects the operation and characteristics of the associated disk device. Set Mode is a logical I/O function and requires the access privilege necessary to perform logical I/O. A single function code is provided:

- IO\$\_SETMODE

This function takes the following device/function dependent argument (other arguments are not valid):

- P1 -- the address of characteristics buffer

## DISK DRIVERS

The P1 argument points to a quadword block shown in Figure 3-3.

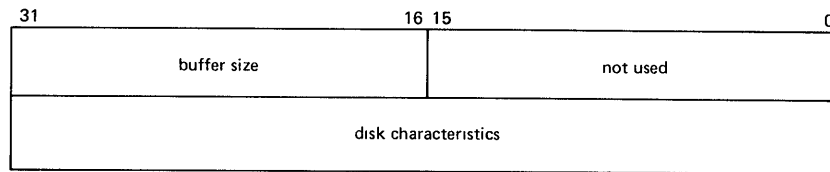


Figure 3-3 Set Mode Characteristics Buffer

The buffer size is the default to be used for disk transfers (this default is normally 512 bytes). Disk characteristics are listed in Table 3-2.

**3.4.3.2 Set Characteristic** - The Set Characteristic function affects the characteristics of the associated disk device. Set Characteristic is a physical I/O function and requires the access privilege necessary to perform physical I/O functions. A single function code is provided:

- IO\$SETCHAR

This function takes the following device/function dependent argument (other arguments are not valid):

- P1 -- the address of characteristics buffer

The P1 argument points to a quadword block as shown in Figure 3-4.

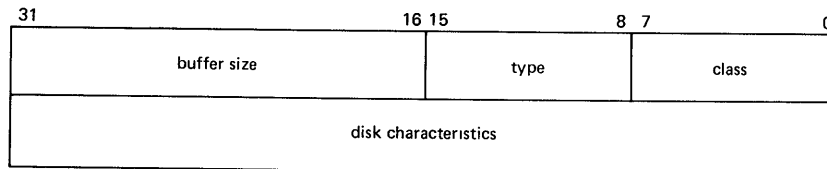


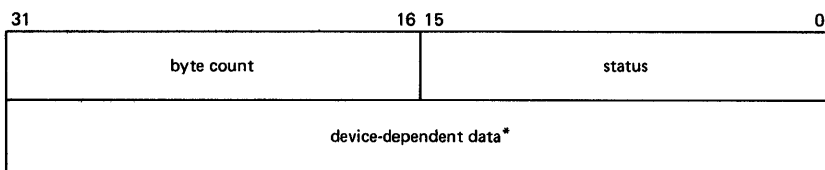
Figure 3-4 Set Characteristic Buffer

The device class for disks is DC\$DISK. Disk types and characteristics are listed in Section 3.3.

## DISK DRIVERS

### 3.5 I/O STATUS BLOCK

Figure 3-5 shows the I/O status block (IOSB) for disk device QIO functions. Table 3-4 lists the status returns for these functions.



\*0 for disk devices

Figure 3-5 IOSB Content

Table 3-4  
Status Returns for Disk Devices

Status	Meaning
SS\$ _NORMAL	Successful completion of the operation specified in the QIO request. The second word of the IOSB can be examined to determine the actual number of bytes transferred to or from the buffer.
SS\$ _CTRLERR	Controller-related error. For example, one or more of the following conditions can cause this error: <ul style="list-style-type: none"> <li>● Late data</li> <li>● Error confirmation</li> <li>● Invalid map register</li> <li>● Interface timeout</li> <li>● Missed transfer</li> <li>● Programming error</li> <li>● Read timeout</li> </ul>
SS\$ _DATACHECK	Data check error. A mismatch between the data in memory and the data on disk was detected during a data check operation (see Section 3.2.1).
SS\$ _DRVERR	Drive-related error. For example, one or more of the following conditions can cause this error: <ul style="list-style-type: none"> <li>● Driver timing error</li> <li>● Illegal function</li> <li>● Illegal register</li> <li>● Operation incomplete</li> <li>● Register modify refused</li> <li>● Write clock failure</li> </ul>

(continued on next page)

## DISK DRIVERS

Table 3-4 (Cont.)  
Status Returns for Disk Devices

Status	Meaning
SS\$ _FORMAT	Format error. Format specified by driver does not correspond to format as specified in sector headers. Disk has been formatted for another computer, such as, DECsystem-20.
SS\$ _IVADDR	Invalid disk address error. Either an invalid starting disk address or a disk address that was incremented causes this error. This error occurs for physical read and write operations or as the result of a hardware error.
SS\$ _MEDOFL	Medium off line. The addressed drive currently does not have a volume mounted and on line.
SS\$ _NONEXDRV	Nonexistent drive. The addressed drive does not exist or the drive select plug has been removed.
SS\$ _PARITY	Parity error. For example, one or more of the following conditions can cause this error: <ul style="list-style-type: none"> <li>● Drive parity error</li> <li>● ECC hard error</li> <li>● Header compare error</li> <li>● Map parity error</li> <li>● Header CRC error</li> <li>● MASSBUS control parity error</li> <li>● MASSBUS data parity error</li> </ul>
SS\$ _UNSAFE	Drive unsafe. The addressed drive is currently unsafe and cannot perform any operation as the result of a hardware error.
SS\$ _VOLINV	Volume invalid. The addressed drive has not been mounted and therefore does not have volume valid set, or a status change has occurred in the drive so that it has changed to an unknown, and therefore, invalid state. All logical and virtual functions will be refected with this status until volume valid is set. Volume valid is set when a volume is mounted and cleared when the volume is unloaded, the respective drive changes to an unknown state, or the power fails. The driver automatically sets volume valid when the proper volume is mounted and/or power is restored.
SS\$ _WASECC	Data check not performed. The function was a read data that was completed successfully by applying one or more ECC corrections. The specified data check, however, was not performed.
SS\$ _WRITLCK	Write lock error. An attempt was made to write on a write locked drive. Volume is hardware write protected.

## DISK DRIVERS

### 3.6 PROGRAMMING EXAMPLE

The following program provides an example of optimizing access time to a disk file. The program creates a file using VAX-11 RMS, stores information concerning the file, and closes the file. The program then accesses the file and reads and writes to the file using the Queue I/O system service.

```

        .TITLE  Disk Driver Programming Example
        .IDENT  /01/

;
; Define necessary symbols
;
        $FIRDEF          ;Define File Information Block Offsets
        $IODEF          ;Define I/O function codes
        $RMSDEF         ;Define RMS-32 Return Status Values

;
; Local storage
;
; Define number of records to be processed
;
NUM_RECS=100                ;One hundred records

;
; Allocate storage for necessary data structures
;
; Allocate File Access Block
;
; A file access block is required by RMS-32 to open and close a file.
;
FAB_BLOCK:
SFAB      ALQ = 100,-          ;Initial file size is to be 100 blocks
          FAC = PUT,-         ;File Access Type is output
          FNA = FILE_NAME,-   ;File name string address
          FNS = FILE_SIZE,-   ;File name string size
          FOP = CTG,-         ;File is to be contiguous
          MRS = 512,-         ;Maximum record size is 512 bytes
          NAM = NAM_BLOCK,-   ;File name block address
          ORG = SEQ,-         ;File organization is to be sequential
          RFM = FIX,-         ;Record format is fixed length

;
; Allocate file information block
;
; A file information block is required as an argument in the Queue I/O
; system service call that accesses a file.
;
FIB_BLOCK:
        .BLKB  FIBSK_LENGTH    ;

;
; Allocate file information block descriptor
;
FIB_DESCR:
        .LONG  FIBSK_LENGTH    ;Length of file information block
        .LONG  FIB_BLOCK       ;Address of file information block

;
; Allocate File Name Block
;
; A file name block is required by RMS-32 to return information concerning
; a file (e.g. the resultant filename string after logical name translation
; and defaults have been applied).
;
NAM_BLOCK:
        SNAM                    ;

;
; Allocate Record Access Block
;
; A record access block is required by RMS-32 for record operations on a
; file.
;
RAB_BLOCK:
SRAB     FAB = FAB_BLOCK,-     ;File access block address
          RAC = SEQ,-          ;Record access is to be sequential
          RBF = RECORD_BUFFER,- ;Record buffer address
          RSZ = 512            ;Record buffer size

;
; Allocate direct access buffer
;
BLOCK_BUFFER:
        .BLKB  1024            ;Direct access buffer is 1024 bytes

;
; Allocate space to store channel number returned by the Assign Channel system
; service

```

## DISK DRIVERS

```

;
DEVICE_CHANNEL:
  .BLKW 1 ;
;
; Allocate device name string and descriptor
;
DEVICE_DESCR:
  .LONG 20$-10$ ; Length of device name string
  .LONG 10$ ; Address of device name string
10$: .ASCII /SYS$DISK/ ; Device on which created file will reside
20$: ; Reference label to calculate length
;
; Allocate file name string and define string length symbol
;
FILE_NAME:
  .ASCII /SYS$DISK:MYDATAFIL.DAT/ ; File name string
FILE_SIZE = .-FILE_NAME ; File name string length
;
; Allocate I/O status quadword storage
;
IO_STATUS:
  .BLKQ 1 ;
;
; Allocate output record buffer
;
RECORD_BUFFER:
  .BLKB 512 ; Record buffer is 512 bytes
;
; Program starting point
;
; The general logic of the program is to create a file called MYDATAFIL.DAT
; using RMS-32, store information concerning the file, write 100 records each
; of which contains its record number in every byte, close the file, and then
; access and read and write the file directly using the Queue I/O system service.
; if any errors are detected, the program returns to its caller with the final
; error status in register R0.
;
  .ENTRY DISK_EXAMPLE, ^M<R2,R3,R4,R5,R6> ; Program starting address
;
; First create the file and open it using RMS-32
;
PART_1:
  SCREATE FAB = FAB_BLOCK ; First part of example
  BLBC R0,20$ ; Create and open file
  ; If low bit clear, creation failure
;
; Second connect the record access block to the created file
;
  SCONNECT RAB = RAB_BLOCK ; Connect the record access block
  BLBC R0,30$ ; If low bit clear, connection failure
;
; Now write 100 records each containing its record number
;
  MOVZBL #NUM_RECS,R6 ; Set record write loop count
;
; Fill each byte of the record to be written with its record number
;
10$: SUBB3 R6,#NUM_RECS+1,R5 ; Calculate record number
  MOVCS #0,(R6),R5,#512,RECORD_BUFFER ; Fill record buffer
;
; Next write the record into the newly created file using RMS-32
;
  SPUT RAB = RAB_BLOCK ; Put record in file
  BLBC R0,30$ ; If low bit clear, put failure
  SOBGTR R6,10$ ; Any more records to write?
;
; The file creation part of the example is almost complete. All that remains to
; be done is to store the file information returned by RMS-32 and close the file.
;
  MOVW NAM_BLOCK+NAMSW_FID,FIB_BLOCK+FIBSW_FID ; Save file identification
  MOVW NAM_BLOCK+NAMSW_FID+2,FIB_BLOCK+FIBSW_FID+2 ; Save sequence number
  MOVW NAM_BLOCK+NAMSW_FID+4,FIB_BLOCK+FIBSW_FID+4 ; Save relative volume
  SCLOSE FAB = FAB_BLOCK ; Close file
  BLBS R0,PART_2 ; If low bit set, successful close
20$: RET ; Return with RMS error status

```

## DISK DRIVERS

```

; Record stream connection or put record failure
; Close file and return status
;
30$:   PUSHL   R0           ;Save error status
       $CLOSE  FAB = FAB_BLOCK ;Close file
       POPL   R0           ;Retrieve error status
       RET     ;Return with RMS error status
;
; The second part of the example illustrates accessing the previously created
; file directly using the Queue I/O system service, randomly reading and writing
; various parts of the file, and then deaccessing the file.
; First assign a channel to the appropriate device and access the file
;
PART_2:
$ASSIGN_S DEVNAM = DEVICE_DESCR, - ;Assign a channel to file device
        CHAN = DEVICE_CHANNEL
        BLBC  R0,20$           ;If low bit clear, assignment failure
        MOVL  #FIBSM_NOWRITE!FIBSM_WRITE,- ;Set for read/write access
        FIB_BLOCK+FIBSL_ACCTL
        $QIOW_S CHAN = DEVICE_CHANNEL, - ;Access file on device channel
        FUNC = #IOS_ACCESS!IOSM_ACCESS,- ;I/O function is access file
        IOSB = IO_STATUS,- ;Address of I/O status quadword
        P1 = FIB_DESCR ;Address of information block descriptor
        BLBC  R0,IOS           ;If low bit clear, access failure
        MOVZWL IO_STATUS,R0 ;Get final I/O completion status
;
10$:   BLBS   R0,30$         ;If low bit set, successful I/O function
       PUSHL   R0           ;Save error status
       $DASSGN_S CHAN = DEVICE_CHANNEL ;Deassign file device channel
       POPL   R0           ;Retrieve error status
20$:   RET     ;Return with I/O error status
;
; The file is now ready to be read and written randomly. Since the records are
; fixed length and exactly one block long, the record number corresponds to the
; virtual block number of the record in the file. Thus a particular record can
; be read or written simply by specifying its record number in the file.
;
; The following code reads 2 records at a time and checks to see that they contain
; their respective record numbers in every byte. The records are then written back
; into the file in reverse order. This results in record 1 having the old contents
; of record 2 and record 2 the old contents of record 1, etc. After the example
; has been run, it is suggested that the file dump utility be used to verify this
; fact.
;
30$:   MOVZBL  #1,R6         ;Set starting record (block) number
;
; Read next 2 records into block buffer
;
40$:   $QIOW_S CHAN = DEVICE_CHANNEL, - ;Read next 2 records from file channel
        FUNC = #IOS_READVBLK,- ;I/O function is read virtual block
        IOSB = IO_STATUS,- ;Address of I/O status quadword
        P1 = BLOCK_BUFFER,- ;Address of I/O buffer
        P2 = #1024,- ;Size of I/O buffer
        P3 = R6 ;Starting virtual block of transfer
        BSBB  50$ ;Check I/O completion status
;
; Check each record to make sure it contains the correct data
;
        SKPC  R6,#512,BLOCK_BUFFER ;Skip over equal record numbers in data
        BNEQ  60$ ;If not equal, data match failure
        ADDL3 #1,R6,R5 ;Calculate even record number
        SKPC  R5,#512,BLOCK_BUFFER+512 ;Skip over equal record numbers in data
        BNEQ  60$ ;If not equal, data match failure
;
; Record data matches
;
; write records in reverse order in file
;
        $QIOW_S CHAN = DEVICE_CHANNEL, - ;write even numbered record in odd slot
        FUNC = #IOS_WRITEVBLK,- ;I/O function is write virtual block
        IOSB = IO_STATUS,- ;Address of I/O status quadword
        P1 = BLOCK_BUFFER+512,- ;Address of even record buffer

```

## DISK DRIVERS

```

                P2 = #512,-           ;Length of even record buffer
                P3 = R6               ;Record number of odd record
BSRB           50$                   ;Check I/O completion status
ADDL3         #1,R6,R5               ;Calculate even record number

$QIOW_S CHAN = DEVICE_CHANNEL,-      ;write odd numbered record in even slot
FUNC = #IOS_WRITEVBLK,-             ;I/O function is write virtual block
IOSB = IO_STATUS,-                  ;Address of I/O status quadword
P1 = BLOCK_BUFFER,-                 ;Address of odd record buffer
P2 = #512,-                           ;Length of odd record buffer
P3 = R5                               ;Record number of even record
BSRB         50$                     ;Check I/O completion status
ACBB         #NUM_RECS-1,#2,R6,40$    ;Any more records to be read?

BRB          70$                     ;

;
; Check I/O completion status
;
50$:  BLBC     R0,70$                  ;If low bit clear, service failure
      MOVZWL  IO_STATUS,R0           ;Get final I/O completion status
      BLBC     R0,70$                  ;if low bit clear, I/O function failure
      RSB

;
; Record number mismatch in data
;
60$:  MNEGL   #4,R0                   ;Set dummy error status value

;
; All records have been read, verified, and odd/even pairs inverted
;
70$:  PUSHL   R0                       ;Save final status
      $QIOW_S CHAN = DEVICE_CHANNEL,- ;Deaccess file
      FUNC = #IOS_DEACCESS             ;I/O function is deaccess file
      $DASSGN_S CHAN = DEVICE_CHANNEL ;Deassign file device channel
      POPL   R0                       ;Retrieve final status
      RET

      .END      DISK_EXAMPLE

```



CHAPTER 4  
MAGNETIC TAPE DRIVER

This chapter describes the use of the VAX/VMS magnetic tape driver. This driver supports the device listed in Table 4-1 and detailed in Section 4.1.1.

Table 4-1  
Magnetic Tape Devices

Model	No. of Tracks	Recording density (bpi)	Tape speed (ips)	Max. data transfer rate in bytes per second	Recording method
TE16	9	800 or 1600	45	36,000 (for 800 bpi); 72,000 (for 1600 bpi)	NRZI or PE <sup>1</sup>

1 NRZI = non-return-to-zero-inverted; PE = phase encoded.

#### 4.1 SUPPORTED MAGNETIC TAPE DEVICES

The following section describes the TE16 magnetic tape drive in greater detail.

##### 4.1.1 TE16 Magnetic Tape Drive

The TE16 magnetic tape drive holds one 9-track reel with a capacity of 40 million characters. The drive reads data at 45 inches per second with an average transfer time of 14 microseconds per byte at the 1600 bpi density.

#### 4.2 DRIVER FEATURES AND CAPABILITIES

The VAX/VMS magnetic tape drivers provide the following features:

- Multiple master adapters and slave formatters
- Different types of devices on a single MASSBUS adapter; for example, RP05 disk and TM03 tape formatter

## MAGNETIC TAPE DRIVER

- Reverse read and reverse data check functions
- Data checks on a per-request, per-file, and/or per-volume basis
- Full recovery from power failure for on-line drives with volumes mounted, including repositioning by the driver
- Extensive error recovery algorithms; for example, non-return-to-zero-inverted (NRZI) error correction
- Logging of device errors in a file that may be displayed by field service or customer personnel
- On-line diagnostic support for drive level diagnostics

The following sections describe master and slave controllers, and data check and error recovery capabilities in greater detail.

### 4.2.1 Master Adapters and Slave Formatters

VAX/VMS supports the use of multiple master adapters of the same type on a system. For example, more than one MASSBUS adapter (MBA) can be used on the same system. A master adapter is a device controller capable of performing and synchronizing data transfers between memory and one or more slave formatters.

VAX/VMS also supports the use of multiple slave formatters per master adapter on a system. For example, more than one TM03 Magnetic Tape Formatter per MBA can be used on a system. A slave formatter accepts data and/or commands from a master adapter and directs the operation of one or more slave drives. The TM03 is a slave formatter. The TE16 Magnetic Tape Transport is a slave drive.

### 4.2.2 Data Check

A data check is made after successful completion of an I/O operation to compare the data in memory with that on the tape. After a write or read (forward) operation, the tape drive backspaces and then performs a write check data operation. After a read in the reverse direction, the tape drive forward spaces and then performs a write check data reverse operation. Magnetic tape drivers support data checks at three levels:

- Per request -- users can specify the data check function modifier (IO\$M\_DATACHECK) on a read logical block, write logical block, read virtual block, write virtual block, read physical block, or write physical block I/O function.
- Per volume -- users can specify the characteristics "data check all reads" and/or "data check all writes" when the volume is mounted. The VAX/VMS Command Language User's Guide describes volume mounting and dismounting.
- Per file -- users can specify the file attributes "data check on read" or "data check on write." File access attributes are specified when the file is accessed. Chapter 9 of this manual and the VAX-11 Record Management Services Reference Manual describe file access.

## MAGNETIC TAPE DRIVER

### 4.2.3 Error Recovery

Error recovery in VAX/VMS is aimed at performing all possible operations to complete an I/O operation successfully. Magnetic tape error recovery operations fall into two categories:

- Handling special conditions such as power failure and interrupt timeout
- Retrying nonfatal controller and/or drive errors

The error recovery algorithm uses a combination of these two types of error recovery operations.

Power failure recovery consists of waiting for mounted drives to be unloaded by the operator. When the drives are reloaded, the driver automatically spaces to the position held before the power failure. The I/O operation that was in progress at the time of the power failure is then re-executed. To solicit reloading of mounted drives, device not ready messages are sent to the operator console after a power failure.

Device timeout is treated as a fatal error with a loss of tape position. A tape on which a timeout has occurred must be dismounted and rewound before the drive position can be established.

Nonfatal controller/drive errors are simply re-executed up to 16 times before returning a fatal error. The tape is repositioned as necessary before each retry.

All normal error recovery (nonspecial conditions) can be inhibited by specifying the inhibit retry function modifier (IO\$M\_INHRETRY). If any error occurs and this modifier is specified, the operation is immediately terminated, and a failure status is returned. This modifier has no effect on power failure and timeout recovery.

Up to 16 extended interrecord gaps can be written during the error recovery for a write operation. The writing of these gaps can be suppressed by specifying the inhibit extended interrecord gap function modifier (IO\$M\_INHEXTGAP).

### 4.3 DEVICE INFORMATION

Users can obtain information on device characteristics by using the \$GETCHN and \$GETDEV system services (see Section 1.10). The information is returned in a user-specified buffer shown in Figure 4-1. Only the first three longwords of the buffer are shown in Figure 4-1 (Figure 1-8 shows the entire buffer).

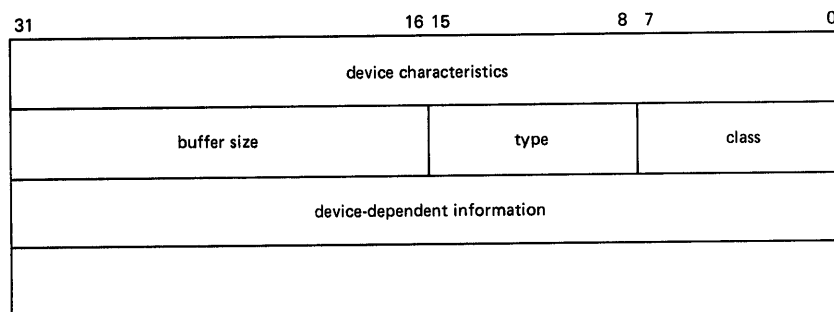


Figure 4-1 Magnetic Tape Information

## MAGNETIC TAPE DRIVER

The device characteristics returned in the first longword are listed in Table 4-2.

Table 4-2  
Magnetic Tape Device-Independent Characteristics

Dynamic Bits <sup>1</sup> (Conditionally Set)	Meaning
DEV\$M_AVL	Device is on line and available
DEV\$M_FOR	Foreign volume
DEV\$M_MNT	Volume mounted
DEV\$M_RCK	Perform data check all reads
DEV\$M_WCK	Perform data check all writes
Static Bits <sup>1</sup> (Always Set)	Meaning
DEV\$M_FOD	File-oriented device
DEV\$M_IDV	Device is capable of input
DEV\$M_ODV	Device is capable of output
DEV\$M_SQD	Device is sequential access

<sup>1</sup> Defined by the \$DEVDEF macro.

The second longword contains information on device class and type, and the buffer size. The device class for tapes is DC\$\_TAPE. The device type is DT\$\_TE16.

The \$DCDEF macro defines the device type and class names. The buffer size is the default to be used for tape transfers (this default is normally 2048 bytes).

The third longword contains device-dependent information. Table 4-3 lists this information. The \$MTDEF macro defines the values listed.

Table 4-3  
Device-Dependent Information for Tape Devices

Value	Meaning
MT\$M_LOST	If set, the current tape position is unknown.
MT\$M_HWL	If set, the selected drive is hardware write-locked.
MT\$M_EOT	If set, an end-of-tape (EOT) condition was encountered by the last operation to move tape in the forward direction.

(Continued on next page)

## MAGNETIC TAPE DRIVER

Table 4-3 (Cont.)  
Device-Dependent Information For Tape Devices

Value	Meaning
MT\$M_EOF	If set, a tape mark was encountered by the last operation to move tape.
MT\$M_BOT	If set, a beginning-of-tape (BOT) marker was encountered by the last operation to move tape in the reverse direction.
MT\$M_PARITY	If set, all data transfers are performed with even parity. If clear (normal case), all data transfers are performed with odd parity. Only NRZI recording at 800 bpi can have even parity.
MT\$V_DENSITY MT\$S_DENSITY	Specifies the density at which all data transfer operations are performed. Possible density values are:  <div style="margin-left: 2em;"> MT\$K_PE_1600    Phase encoded, 1600 bpi.   MT\$K_NRZI_800    Non-return-to-zero-inverted, 800 bpi. </div>
MT\$V_FORMAT MT\$S_FORMAT	Specifies the format in which all data transfers are performed. A possible format value is:  <div style="margin-left: 2em;"> MT\$K_NORMAL11    Normal PDP-11 format. Data bytes are recorded sequentially on tape with each byte occupying exactly one frame. </div>

### 4.4 MAGNETIC TAPE FUNCTION CODES

The VAX/VMS magnetic tape driver can perform logical, virtual, and physical I/O functions.

Logical and physical I/O functions to magnetic tape devices allow sequential access to volume storage and require only that the requesting process have direct access to the device. Virtual I/O functions require intervention by an ancillary control process (ACP) and must be executed in a prescribed order. The normal procedure is to create a file and access it. Information is then written to the file and the file is deaccessed. The file is subsequently accessed, the information is read, and the file is deaccessed. The file can be written over when the information it contains is no longer useful and the file has expired.

Any number of bytes (up to a maximum of 65K) can be read from or written into a single block by a single request. The number of bytes itself has no effect on the applicable quotas (direct I/O, buffered I/O, and AST). Reading or writing any number of bytes subtracts the same amount from a quota.

## MAGNETIC TAPE DRIVER

The volume to which a logical or virtual function is directed must be mounted in order for the function to actually be executed. If it is not, either a device not mounted or invalid volume status is returned in the I/O status block.

Table 4-4 lists the logical, virtual, and physical magnetic tape I/O functions and their function codes. These functions are described in more detail in the following paragraphs and in Appendix A. Chapter 9 describes the QIO level interface to the magnetic tape device ACP.

Table 4-4  
Magnetic Tape I/O Functions

Function Code and Arguments	Type <sup>1</sup>	Function Modifiers	Function
IO\$_CREATE P1,[P2],-[P3],[P4],[P5]	V	IO\$M_CREATE IO\$M_ACCESS	Create a file
IO\$_ACCESS P1,[P2],-[P3],[P4],[P5]	V	IO\$M_CREATE IO\$M_ACCESS	Search a tape for a specified file and access the file if found and IO\$M_ACCESS is set. If the file is not found and IO\$M_CREATE is set, create a file at end-of-tape
IO\$_DEACCESS P1,[P2],-[P3],[P4],[P5]	V		Deaccess a file and, if the file has been written, write out trailer records
IO\$_MODIFY P1,[P2],-[P3],[P4],[P5]	V		Write user labels
IO\$_READVBLK P1,P2	V	IO\$M_DATACHECK IO\$M_INHRETRY IO\$M_REVERSE	Read virtual block
IO\$_READLBLK P1,P2	L	IO\$M_DATACHECK IO\$M_INHRETRY IO\$M_REVERSE	Read logical block
IO\$_READPBLK P1,P2	P	IO\$M_DATACHECK IO\$M_INHRETRY IO\$M_REVERSE	Read physical block
IO\$_WRITEVBLK P1,P2	V	IO\$M_DATACHECK IO\$M_INHRETRY IO\$M_INHEXTGAP	Write virtual block
IO\$_WRITELBLK P1,P2	L	IO\$M_DATACHECK IO\$M_INHRETRY IO\$M_INHEXTGAP	Write logical block

<sup>1</sup> V = virtual; L = logical; P = physical.

(continued on next page)

MAGNETIC TAPE DRIVER

Table 4-4 (Cont.)  
Magnetic Tape I/O Functions

Function Code and Arguments	Type <sup>1</sup>	Function Modifiers	Function
IO\$_WRITEPBLK P1,P2	P	IO\$M_DATACHECK IO\$M_INHRETRY IO\$M_INHEXTGAP	Write physical block
IO\$_REWIND	L	IO\$M_INHRETRY IO\$M_NOWAIT	Reposition tape to the beginning of tape (BOT) marker
IO\$_SKIPFILE P1	L	IO\$M_INHRETRY	Skip past a specified number of tape marks in either a forward or reverse direction
IO\$_SKIPRECORD P1	L	IO\$M_INHRETRY	Skip past a specified number of blocks in either a forward or reverse direction
IO\$_WRITEOF	L	IO\$M_INHRETRY IO\$M_INHEXTGAP	Write an extended interrecord gap followed by a tape mark
IO\$_REWINDOFF	L	IO\$M_INHRETRY IO\$M_NOWAIT	Rewind and unload the tape on the selected drive
IO\$_SENSEMODE	L	IO\$M_INHRETRY	Sense the tape characteristics and return them in the I/O status block
IO\$_SETMODE P1	L	IO\$M_INHRETRY	Set tape characteristics for subsequent operations
IO\$_SETCHAR P1	P	IO\$M_INHRETRY	Set tape characteristics for subsequent operations
IO\$_ACPCONTROL P1,[P2],[-P3],[P4],[P5]	V	IO\$M_DMOUNT	Perform miscellaneous CONTROL FUNCTIONS (SEE SECTION 9.1)
IO\$_MOUNT	V		Mounts a volume; requires mount privilege.

## MAGNETIC TAPE DRIVER

The function-dependent arguments for `IO$_CREATE`, `IO$_ACCESS`, `IO$_DEACCESS`, and `IO$_MODIFY` are:

- P1 -- the address of the File Information Block (FIB) descriptor.
- P2 -- the address of the file name string descriptor (optional). If specified with `IO$_ACCESS`, the name identifies the file being sought. If specified with `IO$_CREATE`, the name is the name of the created file.
- P3 -- the address of the word that is to receive the length of the resultant file name string (optional).
- P4 -- the address of a descriptor for a buffer that is to receive the resultant file name string (optional).
- P5 -- the address of a list of attribute descriptors (optional). If specified with `IO$_ACCESS`, the attributes of the file are returned to the user. If specified with `IO$_CREATE`, P5 is the address of the attribute descriptor list for the new file. All file attributes for `IO$_MODIFY` are ignored.

The function-dependent arguments for `IO$_READVBLK`, `IO$_READLBLK`, `IO$_READPBLK`, `IO$_WRITEVBLK`, `IO$_WRITELBLK`, and `IO$_WRITEPBLK` are:

- P1 -- the starting virtual address of the buffer that is to receive the data in the case of a read operation; or, in the case of a write operation, the virtual address of the buffer that is to be written on the tape.
- P2 -- the number of bytes that are to be read from the tape, or written from memory to the tape.

The function-dependent argument for `IO$_SKIPFILE` and `IO$_SKIPRECORD` is:

- P1 -- the number of tape marks to skip over in the case of a skip file operation; or, in the case of a skip record operation, the number of blocks to skip over. If a positive number is specified, the tape moves forward; if a negative number is specified, the tape moves in reverse. (The maximum number of tape marks or records that P1 can specify is 32,767.)

### 4.4.1 Read

This function reads data into a specified buffer in the forward or reverse direction starting at the next block position.

VAX/VMS provides three read function codes:

- `IO$_READVBLK` - read virtual block
- `IO$_READLBLK` - read logical block
- `IO$_READPBLK` - read physical block

A read virtual block function to a volume that is mounted foreign is converted to a read logical block function. A read virtual block function to a volume that is mounted structured is handled in the normal manner for a file-structured device.

## MAGNETIC TAPE DRIVER

If the reverse function modifier (IO\$M REVERSE) is specified, the read operation is performed in the reverse direction instead of the forward direction.

The data check function modifier (IO\$M DATACHECK) can be used with all read functions. If this modifier is specified, a data check operation is performed after the read data operation has been completed. (A space reverse or space forward is performed between the read and the data check operation.) A data check operation is also performed if the volume read, or the volume on which the file resides (virtual read), has the characteristic "data check all reads." Furthermore, a data check is performed after a virtual read if the file has the attribute "data check on read."

If a read physical block or read logical block operation is performed and the reverse function modifies IO\$M REVERSE is not specified, an end-of-tape status is returned if either of the following conditions occur and no other error condition exists:

- The tape is positioned past the end-of-tape position at the start of the read operation
- The tape enters the end-of-tape region as a result of the read operation

The transferred byte count reflects the actual number of bytes read. If a read in the reverse direction is performed when the tape is positioned past the end-of-tape position, an end-of-tape status is not returned.

If a tape mark is read during a logical or physical read operation in either the forward or reverse direction, an end-of-file status is returned if any of the following conditions exist:

- The tape is positioned past the end-of-tape position at the start of the read operation
- The tape enters the end-of-tape region as a result of the read operation
- A tape mark is read as a result of a read operation but the tape does not enter the end-of-tape region

An end-of-file status is also returned if a read operation in the reverse direction is attempted when the tape is positioned at the BOT marker. All conditions that cause an end-of-file status result in a transferred byte count of zero.

If an attempt is made during a logical or physical read operation to read a block that is larger than the specified memory buffer, a data overrun status is returned. Only the first part of the block is read into the specified buffer. (Only the latter part of the block is read into the buffer on a read in the reverse direction.) The transferred byte count is equal to the actual size of the block. Read reverse starts at the top of the buffer. Thus, the start of the block is at P1 plus P2 minus the length read.

It is not possible to read a block that is less than 14 bytes in length. Such records are termed "noise blocks" and are completely ignored by the driver.

## MAGNETIC TAPE DRIVER

### 4.4.2 Write

This function writes data from a specified buffer to tape in the forward direction starting at the next block position.

VAX/VMS provides three write function codes:

- IO\$\_WRITEVBLK - write virtual block
- IO\$\_WRITELBLK - write logical block
- IO\$\_WRITEPBLK - write physical block

If a write virtual block function is to a volume that is mounted foreign, it is converted to a write logical block function. If a write virtual block function is to a volume that is mounted structured, it is handled in the normal manner for a file-structured device.

The data check function modifier (IO\$\_DATACHECK) can be used with all write functions. If this modifier is specified, a data check operation is performed after the write data operation has been completed. (A space reverse is performed between the write and the data check operation.) A data check operation is also performed if the volume written, or the volume on which the file resides (virtual write), has the characteristic "data check all writes." Furthermore, a data check is performed after a virtual write if the file has the attribute "data check on write."

A data check operation is also forced by the driver when an error occurs during a write operation. This ensures that the data can be reread.

If a write physical block or write logical block operation is performed, an end-of-tape status is returned if either of the following conditions occurs and no other error condition exists:

- The tape is positioned past the end-of-tape position at the start of the write operation
- The tape enters the end-of-tape region as a result of the write operation

(The transferred byte count reflects the size of the block written.)

It is not possible to write a block less than 14 bytes in length. An attempt to do so results in the return of a bad parameter status for the QIO request.

### 4.4.3 Rewind

This function repositions the tape to the beginning-of-tape (BOT) marker. If the IO\$\_NOWAIT function modifier is specified, the I/O operation is completed when the rewind is initiated. Otherwise, I/O completion does not occur until the tape is positioned at the BOT marker. IO\$\_REWIND has no function-dependent arguments.

## MAGNETIC TAPE DRIVER

### 4.4.4 Skip File

This logical I/O function skips past a specified number of tape marks in either a forward or reverse direction. A function-dependent argument (P1) is provided to specify the number of tape marks to be skipped, as shown in Figure 4-2. If a positive file count is specified, the tape moves forward; if a negative file count is specified, the tape moves in reverse. (The actual number of files skipped is returned in the I/O status block.)

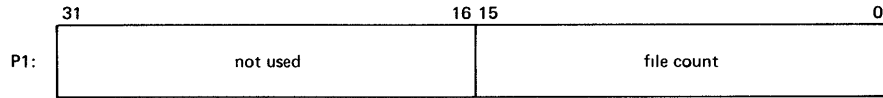


Figure 4-2 IO\$ SKIPFILE Argument

Only tape marks (when the tape moves in either direction) and the BOT marker (when the tape moves in reverse) are counted during a skip file operation. The BOT marker terminates a skip file function in the reverse direction. The end-of-tape (EOT) marker does not terminate a skip file function in either the forward or reverse direction. Note that a negative skip file function leaves the tape positioned just before a tape mark, that is, at the end of a file, unless the BOT marker is encountered whereas, a positive skip file function leaves the tape positioned just past the tape mark.

### 4.4.5 Skip Record

The skip record function skips past a specified number of physical tape blocks in either a forward or reverse direction. A device/function-dependent argument (P1) specifies the number of blocks to skip, as shown in Figure 4-3. If a positive block count is specified, the tape moves forward; if a negative block count is specified, the tape moves in reverse. (The actual number of blocks skipped is returned in the I/O status block.)

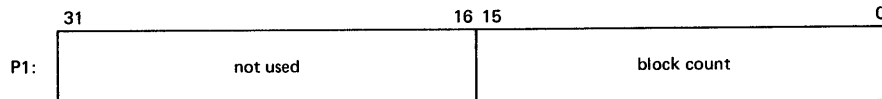


Figure 4-3 IO\$ SKIPRECORD Argument

Skip record is terminated by end-of-file when the tape moves in either direction, by the BOT marker when the tape moves in reverse, and by the EOT marker when the tape moves forward.

### 4.4.6 Write End-of-File

This function writes an extended interrecord gap (of approximately 3 inches for NRZI recording and 1.5 inches for PE recording) followed by a tape mark. No device/function-dependent arguments are used with IO\$ WRITEEOF.

## MAGNETIC TAPE DRIVER

An end-of-tape status is returned in the I/O status block if either of the following conditions is present and no other error conditions occur:

- A write end-of-file function is executed while the tape is positioned past the EOT marker.
- A write end-of-file function causes the tape position to enter the end-of-tape region.

### 4.4.7 Rewind Offline

The rewind offline function rewinds and unloads the tape on the selected drive. If the IO\$M\_NOWAIT function modifier is specified, the I/O operation is completed as soon as the rewind is initiated. No device/function-dependent arguments are used with IO\$\_REWINDOFF.

### 4.4.8 Sense Tape Mode

This function senses the current device-dependent tape characteristics and returns them to the caller in the second longword of the I/O status block (see Table 4-3). The contents of the second longword are identical to the device dependent information shown in Figure 4-1. No device/function-dependent arguments are used with IO\$\_SENSEMODE.

### 4.4.9 Set Mode

Set mode operations affect the operation and characteristics of the associated magnetic tape device. VAX/VMS defines two types of set mode functions:

- Set Mode
- Set Characteristic

4.4.9.1 Set Mode - The Set Mode function affects the characteristics of the associated tape device. Set Mode is a logical I/O function and requires the access privilege necessary to perform logical I/O. A single function code is provided:

- IO\$\_SETMODE

This function takes the following device/function-dependent argument (other arguments are ignored):

- P1 -- the address of a quadword characteristics buffer

Figure 4-4 shows the quadword Set Mode characteristics buffer.

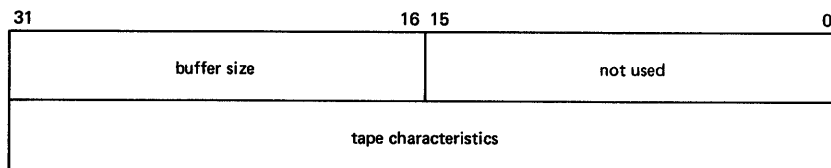


Figure 4-4 Set Mode Characteristics Buffer

## MAGNETIC TAPE DRIVER

Table 4-5 lists the tape characteristics and their meanings. The \$MTDEF macro defines the symbols listed.

Table 4-5  
Set Mode and Set Characteristic Magnetic Tape Characteristics

<p>MT\$M_PARITY</p>	<p>If set, all data transfers are performed with even parity. If clear (normal case), all data transfers are performed with odd parity. Even parity can be selected only for NRZI recording at 800 bpi. Even parity cannot be selected for phase encoded recording (tape density is MT\$K_PE_1600) and is ignored.</p>
<p>MT\$V_DENSITY MT\$S_DENSITY</p>	<p>Specifies the density at which all data transfers are performed. Tape density can be set only when the selected drive's tape position is at the BOT marker. Possible density values are:</p> <p>MT\$K_DEFAULT      Default system density</p> <p>MT\$K_PE_1600      Phase encoded, 1600 bpi</p> <p>MT\$K_NRZI_800      Non-return-to-zero-inverted,      800 bpi</p>
<p>MT\$V_FORMAT MT\$S_FORMAT</p>	<p>Specifies the format in which all data transfers are performed. Possible format values are:</p> <p>MT\$K_DEFAULT      Default system format</p> <p>MT\$K_NORMAL11      Normal PDP-11 format. Data bytes are recorded sequentially on tape with each byte occupying exactly one frame</p>

4.4.9.2 **Set Characteristic** - The Set Characteristic function also affects the characteristics of the associated tape device. Set Characteristic is a physical I/O function and requires the access privilege necessary to perform physical I/O functions. A single function code is provided:

- IO\$\_SETCHAR

This function takes the following device/function-dependent argument (other arguments are not valid):

- P1 -- the address of a quadword characteristics buffer

Figure 4-5 shows the quadword Set Characteristic characteristics buffer.

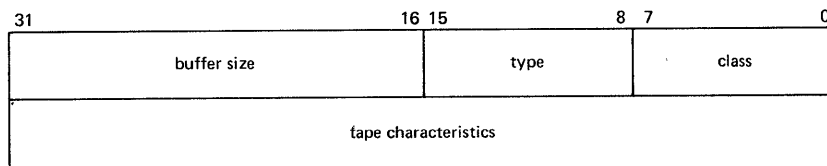


Figure 4-5 Set Characteristic Buffer

## MAGNETIC TAPE DRIVER

The first longword contains information on device class and type, and the buffer size. The device class for tapes is DC\$\_TAPE. The device type is DT\$\_TE16.

The \$DCDEF macro defines the device type and class names. The buffer size is the default to be used for tape transfers (this default is normally 2048 bytes).

Table 4-5 lists the tape characteristics for the Set Characteristic function.

### 4.5 I/O STATUS BLOCK

The I/O status block (IOSB) for QIO functions on magnetic tape devices is shown in Figure 4-6. Table 4-6 lists the status returns for these functions. Table 4-3 (in Section 4.3) lists the device-dependent data returned in the second longword. The IO\$\_SENSEMODE function can be used to return this data.

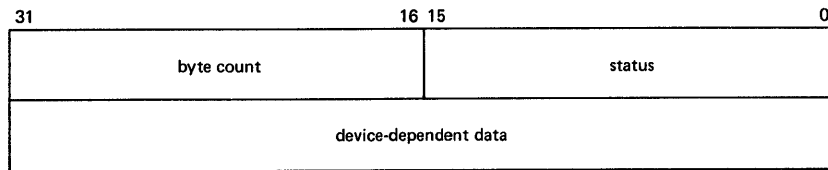


Figure 4-6 IOSB Content

The byte count is the actual number of bytes transferred to or from the process buffer or the number of files or blocks skipped.

Table 4-6  
Status Returns for Tape Devices

Status	Meaning
SS\$_NORMAL	Successful completion of the operation specified in the QIO request. The second word of the IOSB can be examined to determine the actual number of bytes transferred to or from the buffer or the number of files or blocks skipped.
SS\$_CTRLERR	<p>Controller-related error. One or more of the following conditions can cause this error:</p> <ul style="list-style-type: none"> <li>● Data late</li> <li>● Error confirmation</li> <li>● Invalid map register</li> <li>● Interface timeout</li> <li>● Missed transfer</li> <li>● Programming error</li> <li>● Read timeout</li> </ul>

(continued on next page)

MAGNETIC TAPE DRIVER

Table 4-6 (Cont.)  
Status Returns for Tape Devices

Status	Meaning
SS\$_DATACHECK	Write check error. A mismatch between the data in memory and the data on tape was detected during a write check operation. (See Section 4.2.1)
SS\$_DRVERR	<p>Drive-related error. One or more of the following conditions can cause this error:</p> <ul style="list-style-type: none"> <li>● Drive timing error</li> <li>● Illegal function</li> <li>● Illegal register</li> <li>● Operation incomplete</li> <li>● Register modify refused</li> <li>● Nonexecutable function</li> </ul>
SS\$_ENDOFFILE	End-of-file condition. A tape mark was encountered during the operation. For data transfer functions, the byte count is 0; for skip record functions, the count is the number of blocks skipped.
SS\$_ENDOFTAPE	End-of-tape condition. This is a normal completion and is typically treated as such. The end of an input tape is normally denoted by a series of tape marks and/or trailer labels that are independent of the end-of-tape marker. This ensures a decision based on a consistent end-of-tape condition independent of the tape drive.
SS\$_FORMAT	Format error. Format specified by last set tape characteristics function is not implemented in slave controller.
SS\$_MEDOFL	Medium offline. The addressed drive currently does not have a volume mounted and on line.
SS\$_NONEXDRV	Nonexistent drive. The addressed drive does not exist.

(continued on next page)

MAGNETIC TAPE DRIVER

Table 4-6 (Cont.)  
Status Returns for Tape Devices

Status	Meaning
SS\$ _PARITY	<p>Parity error. One or more of the following conditions can cause this error:</p> <ul style="list-style-type: none"> <li>• CRC error (NRZI only)</li> <li>• Control bus parity error</li> <li>• Correctable data error (PE only)</li> <li>• Correctable skew (PE only)</li> <li>• Data bus parity error</li> <li>• Incorrectable error (PE only)</li> <li>• Invalid tape mark (NRZI only)</li> <li>• Nonstandard gap</li> <li>• Longitudinal parity error (NRZI only)</li> <li>• Format error (PE only)</li> <li>• Vertical parity error (NRZI only)</li> <li>• Map parity error</li> <li>• MASSBUS control parity error</li> <li>• MASSBUS data parity error</li> <li>• Read data substitute</li> </ul>
SS\$ _UNSAFE	<p>Drive unsafe. The addressed drive is currently unsafe and cannot perform any function.</p>
SS\$ _VOLINV	<p>Volume invalid. The addressed drive has not been mounted and therefore does not have volume valid set, or a status change has occurred in the drive so that it has changed to an unknown, and therefore, invalid state. All logical and virtual functions will be rejected with this status until volume valid is set. Volume valid is set when a volume is mounted and cleared when the volume is unloaded, the respective drive changes to an unknown state, or the power fails. The driver automatically sets volume valid when the proper volume is mounted and/or power is restored.</p>
SS\$ _WRITLCK	<p>Write lock error. An attempt was made to write on a write locked drive.</p>
SS\$ _DATAOVERUN	<p>Data overrun. The data block read was longer than the assigned buffer. In the case of a read reverse, the last data on tape (that is, the data nearest the end-of-tape at the beginning of the operation) is the first data read. This data is in the buffer.</p>

## MAGNETIC TAPE DRIVER

### 4.6 PROGRAMMING EXAMPLE

The following program is an example of how data is written to and read from magnetic tape. In the example, QIO operations are performed through the magnetic tape ACP. These operations could have been performed directly on the device using the magnetic tape driver. However, this would have involved additional programming, for example, writing header labels and trailer labels.

```

        .TITLE MAGTAPE PROGRAMMING EXAMPLE
        .IDENT /01/

;
; Define necessary symbols
;
        SFIBDEF                ;Define file information block symbols
        SIODEF                 ;Define I/O function codes

;
; Allocate storage for the necessary data structures
;
;
; Allocate magtape device name string and descriptor
;
TAPENAME:                ;
        .LONG    20$-10$      ;Length of name string
        .LONG    10$         ;Address of name string
10$:      .ASCII  /TAPE/      ;Name string
20$:      ;Reference label

;
; Allocate space to store assigned channel number
;
TAPECHAN:                ;
        .BLKW   1             ;Tape channel number

;
; Allocate space for the I/O status quadword
;
IOSTATUS:                ;
        .BLKQ   1             ;I/O status quadword

;
; Allocate storage for the input/output buffer
;
BUFFER:
        .REPT   256
        .ASCII  /A/           ;Initialise buffer to contain 'A'
        .ENDR

; We now define the FIB-file information block-which the ACP uses
; in order to access,deaccess the file.We supply some information
; in this block and the ACP will supply further information.
;
FIB_DESCR:
        .LONG    ENDFIB-FIB   ;Start of FIB
        .LONG    FIB          ;Length of file information block
FIB:     .LONG    FIB$M_WRITE!FIB$M_NOWRITE ;Read/write access allowed
        .WORD    0,0,0        ;File ID
        .WORD    0,0,0        ;Directory ID
        .LONG    0            ;Context
        .WORD    0            ;Name flags
        .WORD    0            ;Extend control
ENDFIB:  ;Reference label

;
; we now define the file name string and descriptor
;
NAME_DESCR:
        .LONG    END_NAME-NAME ;File name descriptor
        .LONG    NAME          ;Address of name string
NAME:    .ASCII  "MYDATA.DAT;1" ;File name string
END_NAME: ;Reference label

;
; Now the main program
;
; The program firstly assigns a channel to the magnetic tape unit.
; It then performs an access function to create and access a file
; called "MYDATA.DAT". It now writes 26 blocks of data to the tape
; containing the letters of the alphabet. The first block contains
; all A's the next all B's and so on. It starts by writing a block
; of 256 bytes and each subsequent block is reduced in size by two
; bytes so by the time it writes the block containing Z's the block
; size is only 206 bytes. The magtape ACP will not allow reading of
; a file that has been written until one of three things happens.
; The file is de-accessed, the file is rewound or the file is back-
; spaced. In this example the file is backspaced zero blocks and
; then it is read in reverse (incrementing the block size every block
; and the data checked against what is meant to be there. If all is
; well the file is de-accessed and the program exits
;

```

# MAGNETIC TAPE DRIVER

```

ENTRY  MAGTAPE_EXAMPLE, ^M<R3,R4,R5,R6,R7,R8>

;
; First assign a channel to the tape unit
;
      SASSIGN_S TAPENAME,TAPECHAN      ;Assign tape unit
      CMPW      #SSS_NORMAL,R0        ;OK?
      BSBW      ERRCHECK              ;Find out

;
; Next create and access the file 'MYDATA.DAT'
;

      SQIOW_S  CHAN=TAPECHAN,-        ;Channel is magtape
                FUNC=#IOS_CREATE!IOSM_ACCESS!IOSM_CREATE,- ;Function is create
                IOSB=IOSTATUS,-      ;Address of I/O status word
                P1=FIB_DESCR,-       ;FIB descriptor
                P2=#NAME_DESCR,-     ;Name descriptor
      CMPW      #SSS_NORMAL,R0        ;OK?
      BSBW      ERRCHECK              ;Find out

;
; LOOP1 consists of writing the alphabet to the tape as described earlier
;

      MOVL      #26,R5                 ;Set up loop count
      MOVL      #256,R3                ;Set up initial byte count in R3
LOOP1:  SQIOW_S  CHAN=TAPECHAN,-        ;Perform QIO to tape channel
                FUNC=#IOS_WRITEVBLK,- ;Function is write virtual block
                P1=BUFFER,-           ;Buffer address
                P2=R3                 ;Byte count
      CMPW      #SSS_NORMAL,R0        ;OK?
      BSBW      ERRCHECK              ;Find out

;
; Now we decrement the byte count ready for the next write, set up a
; loop count for updating the character and LOOP2 performs the update
;

      SUBL2     #2,R3                  ;Decrement byte count for next write
      MOVL      R3,R8                  ;Copy byte count to R8 for LOOP2 count
      MOVAL     BUFFER,R7              ;Get buffer address in R7
LOOP2:  INCB     (R7)+                  ;Increment character
        SOBGTR  R8,LOOP2              ;Until finished
        SOBGTR  R5,LOOP1              ;Repeat LOOP1 until alphabet complete

;
; We now fall through LOOP1 and should update the byte count so that
; it truly reflects the size of the last block written to the tape
;

      ADDL2     #2,R3                  ;Update byte count

;
; We now want to read the tape but must first perform one of the three
; operations outlined above otherwise the ACP will not allow write
; access. We will perform an ACP control function on it specifying
; skip zero blocks. This is a special case of skip reverse and will
; cause the ACP to now allow read access.
;

      CLRL      FIB+FIBSLs_CNTRLVAL   ;Set up to space zero blocks
      MOVW      #FIBsC_SPACE,FIB+FIBsCNTRLFUNC ;Set up for space function
      SQIOW_S  CHAN=TAPECHAN,-        ;Perform QIO to tape channel
                FUNC=#IOS_ACPCONTROL,- ;Perform an ACP control function
                P1=FIB_DESCR          ;Define the FIB
      CMPW      #SSS_NORMAL,R0        ;Success?
      BSBW      ERRCHECK              ;Find out

;
; Now we read the file in reverse
;

      MOVL      #26,R5                 ;Set up loop count
      MOVW      #^A/Z7,R6             ;Get first character in R6
LOOP3:  MOVAL     BUFFER,R7              ;And buffer address to R7
        SQIOW_S  CHAN=TAPECHAN,-        ;Channel is magtape
                FUNC=#IOS_READVBLK!IOSM_REVERSE,- ;Function is read reverse
                IOSB=IOSTATUS,-      ;Define I/O status quadword
                P1=BUFFER,-           ;And buffer address
                P2=R3                 ;R3 bytes
      CMPW      #SSS_NORMAL,R0        ;Success?
      BSBW      ERRCHECK              ;Find out

;
; Now we will check the data we have read in to make sure
; that it agrees with what was written
;

CHECKDATA: MOVL      R3,R4              ;Copy R3 to R4 for loop count
           CMPB     (R7)+,R6            ;Check each character
           BNEQ     MISMATCH            ;Print message on error
           SOBGTR  R4,CHECKDATA         ;Carry on until finished
           DECB     R6                  ;Go backwards through alphabet
           ADDL2     #2,R3              ;Update byte count by 2 for next block
           SOBGTR  R5,LOOP3            ;Read next block

```

# MAGNETIC TAPE DRIVER

```
;/ Now we deaccess the file
;/
      $QIOW_S CHAN=TAPECHAN,-      ;Channel is magtape
              FUNC=#IOS_DEACCESS,- ;Deaccess function
              IOSB=IOSTATUS        ;I/O status

;/ Now we deassign the channel and exit
;/
EXIT:  $DASSGN_S CHAN=TAPECHAN      ;Deassign channel
      RET                          ;Exit

;/ we are now at a place where normally we would attempt to generate some error
;/ message but for this example we will simply exit
;/
MISMATCH:
      BRB      EXIT                ;Exit
ERRCHECK:
      RNEQ     EXIT                ;If error then exit
      RSB      EXIT                ;Exit if not OK
      RSB      EXIT                ;Else return
.END   MAGTAPE_EXAMPLE
```



## CHAPTER 5

### LINE PRINTER DRIVER

This chapter describes the use of the VAX/VMS line printer driver. This driver supports the LP11 Line Printer Interface and the LAll DECprinter I.

#### 5.1 SUPPORTED LINE PRINTER DEVICES

The following sections describe the LP11 Line Printer Interface and the LAll DECprinter I.

##### 5.1.1 LP11 Line Printer Interface

The LP11 is a high-speed, 132-column, line printer available in several models. Printers are available with either a 64- or 96-character ASCII print set. The LP11-R and LP11-S are fully buffered models that operate at a standard speed of 1110 lines per minute. Other LP11 models have 20-character print buffers, and can print at full speed if the printed line is 20 characters or less. Longer lines are printed at a slower rate. Forms with up to six parts can be used for multiple copies.

##### 5.1.2 LAll DECprinter I

The LAll DECprinter I is a medium-speed printer that operates at a standard speed of 180 characters per second. It incorporates such features as a forms length switch to set the top of form to any of 11 common lengths, paper-out switch and alarm, and variable forms width. The LAll uses a 96-character ASCII set; the column width is 132 characters.

#### 5.2 DRIVER FEATURES AND CAPABILITIES

The VAX/VMS line printer driver provides output character formatting and error recovery, as described in the following sections.

## LINE PRINTER DRIVER

### 5.2.1 Output Character Formatting

In write virtual and write logical block operations, user-supplied characters are output as follows (write physical block data is not formatted, but output directly):

1. Rubouts are discarded.
2. Tabs move the horizontal print position to the next MODULO (8) position.
3. All lowercase alphabetic characters are converted to uppercase before printing (unless the characteristic specifying lowercase characters is set; see Section 5.4.2 and Table 5-2).
4. On printers where the line feed, form feed, vertical tab, and return characters empty the printer buffer, returns are held back and output only if the next character is not a form feed, line feed, or vertical tab. Returns are always output on units that do not have the return function characteristic set (see Section 5.4.2 and Table 5-2).
5. The horizontal print position is incremented on the output of all nonprinting characters such as the space character. Nonprinting characters are discarded if the horizontal print position is equal to or greater than the carriage width.
6. On printers without mechanical form feed (the form feed function characteristic is not set; see Section 5.4.2 and Table 5-2), a form feed is converted to multiple line feeds. The number of line feeds is based on the current line count and the page length.
7. Print lines are counted and returned to the caller in the second longword of the I/O status block.

### 5.2.2 Error Recovery

The VAX/VMS line printer driver performs the following error recovery operations:

- If the printer is off line for 30 seconds, a "device not ready" message is sent to the system operator process.
- If the printer runs out of paper or has a fault condition, a "device not ready" message is sent to the system operator every 30 seconds.
- The current operation is retried every 2 seconds to test for a changed situation, for example, the printer coming on line.
- The current I/O operation can be canceled at the next timeout without the printer being on line.
- When the printer comes on line, device operation resumes automatically.

## LINE PRINTER DRIVER

### 5.3 DEVICE INFORMATION

The user process can obtain information on printer characteristics by using the \$GETCHN and \$GETDEV system services (see Section 1.10). The printer-specific information is returned in the first three longwords of a user-specified buffer, as shown in Figure 5-1 (Figure 1-8 shows the entire buffer).

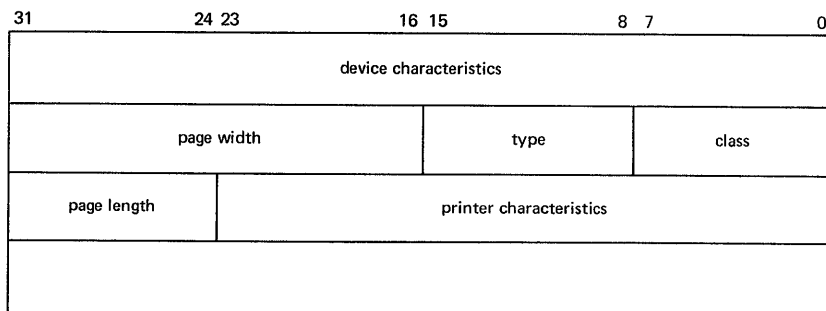


Figure 5-1 Printer Information

The first longword contains device-independent data. The second and third longwords contain device-dependent data.

Table 5-1 lists the device-independent characteristics returned in the first longword.

Table 5-1  
Printer Device-Independent Characteristics

Dynamic Bits <sup>1</sup> (Conditionally Set)	Meaning
DEV\$M_SPL	Spooled device
DEV\$M_AVL	Printer is on line and available
Static Bits <sup>1</sup> (Always Set)	Meaning
DEV\$M_REC	Record-oriented device
DEV\$M_CCL	Carriage control
DEV\$M_ODV	Device is capable of output

<sup>1</sup> Defined by the \$DEVDEF macro.

In the second longword, the device class is DC\$\_LP. The printer type is a value that corresponds to the printer: LP\$\_LP11 or LP\$\_L11. The page width is a value in the range of 0 to 255.

The third longword contains printer characteristics and the page length. The printer characteristics part can contain any of the values listed in Table 5-2.

c

**LINE PRINTER DRIVER**

Table 5-2  
Printer Device-Dependent Characteristics

Value	Meaning
LP\$M_LOWER	Printer can print lowercase characters. If this value is not set, all lowercase characters are converted to uppercase when output.
LP\$M_MECHFORM	Printer has mechanical form feed. This characteristic is used when variable form length is required, for example, check printing. Driver sends ASCII form feed (decimal 12). Otherwise, multiple line feeds are generated. The page length determines the number of line feeds.
LP\$M_CR	Printer requires carriage return. (See note 4, Section 5.2.1).

Maximum page length is 255.

The \$LPDEF macro defines the values for the printer characteristics; the \$DCDEF macro defines the device class and types.

#### 5.4 LINE PRINTER FUNCTION CODES

The basic line printer I/O functions are write, sense mode, and set mode. None of the function codes takes function modifiers.

##### 5.4.1 Write

The line printer write functions print the contents of the user buffer on the designated printer.

The write functions and their QIO function codes are:

- IO\$\_WRITEVBLK - write virtual block
- IO\$\_WRITELBLK - write logical block
- IO\$\_WRITEPBLK - write physical block (the data is not formatted, but output directly, as in PASSALL mode on terminals)

The write function codes can take the following device/function dependent arguments:

- P1 = the starting virtual address of the buffer that is to be written
- P2 = the number of bytes that are to be written

## LINE PRINTER DRIVER

- P3 (ignored)
- P4 = carriage control specifier except for write physical block operations (write function carriage control is described in Section 5.4.1.1)

P3, P5, and P6 are not meaningful for line printer write operations.

In write virtual block and write logical block operations, the buffer specified by P1 and P2 is formatted for the selected line printer and includes the carriage control information specified by P4.

All lowercase characters are converted to uppercase if the characteristics of the selected terminal do not include LP\$M\_LOWER (this does not apply to write physical block operations).

Multiple line feeds are generated for form feeds only if the printer does not have a mechanical form feed, that is, the LP\$M\_MECHFORM characteristic. The number of line feeds generated depends on the current page position and the length of the page.

Section 5.2.1 describes character formatting in greater detail.

**5.4.1.1 Write Function Carriage Control** - The P4 argument is a longword that specifies carriage control. Carriage control determines the next printing position on the line printer. (P4 is ignored in a write physical block operation.) Figure 5-2 shows the P4 longword format.

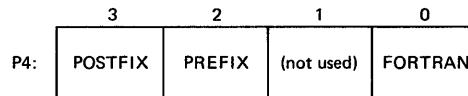


Figure 5-2 P4 Carriage Control Specifier

Only bytes 0, 2, and 3 in the longword are used. Byte 1 is ignored. If the low-order byte (byte 0) is not 0, the contents of the longword are interpreted as a FORTRAN carriage control specifier. Table 5-3 lists the possible byte 0 values (in hexadecimal) and their meanings.

If the low-order byte (byte 0) is 0, bytes 2 and 3 of the P4 longword are interpreted as the prefix and postfix carriage control specifiers. The prefix (byte 2) specifies the carriage control before the buffer contents are printed. The postfix (byte 3) specifies the carriage control after the buffer contents are printed. The sequence is:

Prefix carriage control - Print - Postfix carriage control

The prefix and postfix bytes, although interpreted separately, use the same encoding scheme. Table 5-4 shows this encoding scheme in hexadecimal.

LINE PRINTER DRIVER

Table 5-3  
Write Function Carriage Control (FORTRAN: Byte 0 not equal to 0)

Byte 0 Value (hexadecimal)	ASCII Character	Meaning
20	(space)	Single-space carriage control. (Sequence: line feed, print buffer contents, return.)
30	0	Double-space carriage control. (Sequence: line feed, line feed print buffer contents, return.)
31	1	Page eject carriage control. (Sequence: form feed, print buffer contents, return.)
2B	+	Overprint carriage control. (Sequence: print buffer contents, return.) Allows double printing for emphasis.
24	\$	Prompt carriage control. (Sequence: line feed, print buffer contents.)
All other values		Same as ASCII space character: single-space carriage control.

Table 5-4  
Write Function Carriage Control (P4 byte 0 equal to 0)

Prefix/Postfix Bytes  
(Hexadecimal)

Bit 7	Bits 0 - 6			Meaning
0	0			No carriage control is specified, that is, NULL.
0	1-7F			Bits 0 through 6 are a count of line feeds.
Bit 7	Bit 6	Bit 5	Bits 0-4	Meaning
1	0	0	1-1F	Output the single ASCII control character specified by the configuration of bits 0 through 4 (7-bit character set).
1	1	0	1-1F	Output the single ASCII control character specified by the configuration of bits 0 through 4 which are translated as ASCII characters 128 through 159 (8-bit character set).

## LINE PRINTER DRIVER

Figure 5-3 shows the prefix and postfix hexadecimal coding that produces the carriage control functions listed in Table 5-3. Except for the last example (line skipping), this is an alternative way to achieve these controls.

	(Space)			
P4:		8D	1	-
		0		
		Sequence:		
		Prefix = NL		
		Print		
		Postfix = CR		
		"0"		
P4:		8D	2	-
		0		
		Sequence:		
		Prefix = LF, LF		
		Print		
		Postfix = CR		
		"1"		
P4:		8D	8C	-
		0		
		Sequence:		
		Prefix = FF		
		Print		
		Postfix = CR		
		"+"		
P4:		8D	0	-
		0		
		Sequence:		
		Prefix = NULL		
		Print		
		Postfix = CR		
		"\$"		
P4:		0	8A	-
		0		
		Sequence:		
		Prefix = LF		
		Print		
		Postfix = NULL		
		Example: Skip 24 lines before printing		
P4:		8D	18	-
		0		
		Sequence:		
		Prefix = 24LF		
		Print		
		Postfix = CR		

Figure 5-3 Write Function Carriage Control  
(Prefix and Postfix Coding)

In the first example, the prefix/postfix coding for a single-space carriage control (line feed, print buffer contents, return) is obtained by placing the value (1) in the second (prefix) byte and the sum of the bit 7 value (80) and the return value (D) in the third (postfix) byte:

$$\begin{array}{r}
 80 \text{ (bit 7 = 1)} \\
 + \text{ D (return)} \\
 \hline
 8D \text{ (postfix = return)}
 \end{array}$$

## LINE PRINTER DRIVER

### 5.4.2 Sense Printer Mode

This function senses the current device-dependent printer characteristics and returns them in the second longword of the I/O status block. No device/function-dependent arguments are used with IO\$\_SENSEMODE.

### 5.4.3 Set Mode

Set mode operations affect the operation and characteristics of the associated line printer. VAX/VMS provides two types of set mode functions: Set Mode and Set characteristics. Set Mode requires logical I/O privilege. Set characteristics requires physical I/O privilege. Two function codes are provided:

- IO\$\_SETMODE
- IO\$\_SETCHAR

These functions take the following device/function-dependent argument (other arguments are not valid):

- P1 -- the address of a characteristics buffer

Figure 5-4 shows the quadword P1 characteristics buffer for IO\$\_SETMODE. Figure 5-5 shows the same buffer for IO\$\_SETCHAR.

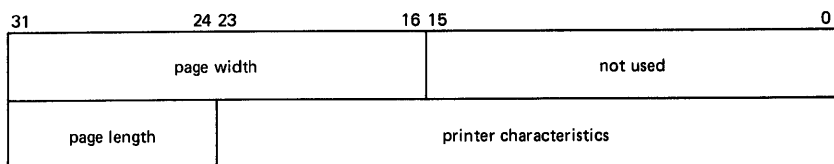


Figure 5-4 Set Mode Characteristics Buffer

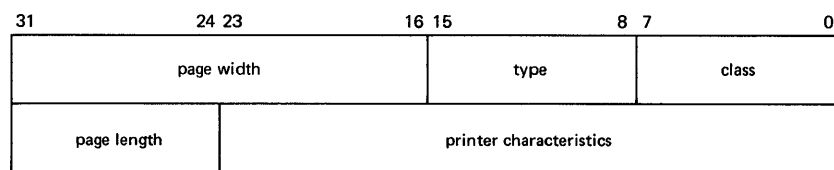


Figure 5-5 Set Characteristic Characteristics Buffer

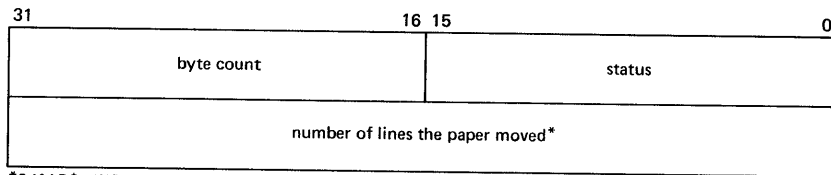
In the buffer, the device class is DC\$\_LP. The printer type is a value that corresponds to the printer: DT\$\_LP11 or DT\$\_L11. The type can be changed by the IO\$\_SETCHAR function. The page width is a value in the range of 0 to 255.

The printer characteristics part of the buffer can contain any of the values listed in Table 5-2.

## LINE PRINTER DRIVER

### 5.5 I/O STATUS BLOCK

The I/O status blocks (IOSB) for the write and set mode I/O functions are shown in Figure 5-6 and 5-7. Table 5-5 lists the status returns for these functions.



\*0 if IOSB\_WRITEPBLK

Figure 5-6 IOSB Contents - Write Function

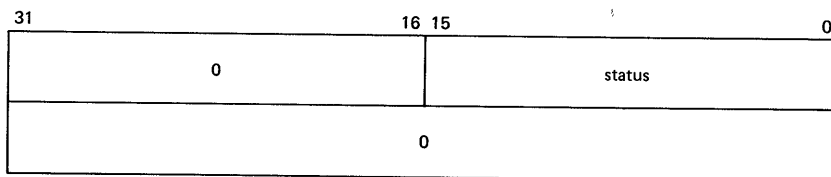


Figure 5-7 IOSB Contents - Set Mode Function

Table 5-5  
Line Printer QIO Status Returns

Status	Meaning
SS\$ _NORMAL	Successful completion. The operation specified in the QIO was completed successfully. On a write operation, the second word of the IOSB can be examined to determine the number of bytes written.
SS\$ _ABORT	The operation was canceled by the Cancel I/O on Channel (\$CANCEL) system service.

## LINE PRINTER DRIVER

### 5.6 PROGRAMMING EXAMPLE

The following simple program is an example of I/O to the line printer that shows how to use the different carriage control formats. This program prints out the contents of the output buffer (OUT\_BUFFER) 10 times using 10 different carriage control formats. The formats are held in location OUTPUT\_FORMAT.

```

        .TITLE  LINE PRINTER PROGRAMMING EXAMPLE
        .IDENT  /01/
;Define necessary symbols
;
        $IODEF                                ;Define I/O function codes
;
; Allocate storage for the necessary data structures
; Allocate output buffer and fill with required output text
;
OUT_BUFFER:
        .ASCII  "VAX_PRINTER_EXAMPLE"
OUT_BUFFER_SIZE=.OUT_BUFFER                ;Define size of output string
;
; Allocate device name string and descriptor
;
DEVICE_DESCR:
        .LONG   20$-10$                      ;Length of name string
        .LONG   10$                          ;Address of name string
10$:    .ASCII  /LINE_PRINTER/              ;Name string of output device
20$:    .BYTE   36,0,0,0                     ;Reference label to calculate length
;
; Allocate space to store assigned channel number
;
DEVICE_CHANNEL:
        .BLKW   1                            ;Channel number
;
; Now set up the carriage control formats
;
OUTPUT_FORMAT:
        .BYTE   0,0,0,0                      ;No carriage control
        .BYTE   32,0,0,0                     ;Blank=LF+...TEXT...+CR
        .BYTE   48,0,0,0                     ;Zero=LF+LF+...TEXT...+CR
        .BYTE   49,0,0,0                     ;One=FF+...TEXT...+CR
        .BYTE   43,0,0,0                     ;Plus=Overprint...+CR
        .BYTE   36,0,0,0                     ;Dollar=LF+TEXT(Prompt)
;
; Now the prefix-postfix carriage control formats
;
        .BYTE   0,0,1,141                    ;LF+...TEXT...+CR
        .BYTE   0,0,24,141                   ;24LF+...TEXT...+CR
        .BYTE   0,0,2,141                    ;LF+LF+...TEXT...+CR
        .BYTE   0,0,140,141                  ;FF+...TEXT...+CR
;
; Program starting point
;
; The program assigns a channel to the output device, sets up a loop
; Count for the number of times it wishes to print, and performs ten
; QIO and wait system services. The channel is then deassigned.
;
        .ENTRY  PRINTER_EXAMPLE,"M<R2,R3>;Program starting address
;
; First assign a channel to the output device
;
        $ASSIGN_S DEVNAM=DEVICE_DESCR,-      ;Assign a channel to printer
        CHAN=DEVICE_CHANNEL
        BLBC    R0,50$                       ;If low bit clear, assignment failure
        MOVL   #11,R3                        ;Set up loop count
        MOVAL  OUTPUT_FORMAT,R2             ;Set up o/p format address in R2
;
; Start of printing loop
;
30$:    $QIOW_S CHAN=DEVICE_CHANNEL,-        ;Print on device channel
        FUNC=#IOS_WRITEVBLK,-              ;I/O function is write virtual
        P1=OUT_BUFFER,-                    ;Address of output buffer
        P2=#OUT_BUFFER_SIZE,-              ;Size of buffer to print
        P4=(R2)+                            ;Format control in R2
        ;will auto-increment.
        BLBC    R0,40$                       ;If low bit clear, i/o failure
        SOBGTR R3,30$                        ;Branch if not finished
40$:    $DASSGN_S CHAN=DEVICE_CHANNEL        ;Deassign channel
50$:    RET                                  ;Return
        .END    PRINTER_EXAMPLE

```

## CHAPTER 6

### CARD READER DRIVER

This chapter describes the use of the VAX/VMS card reader driver. This driver supports the CR11 Card Reader.

#### 6.1 SUPPORTED CARD READER DEVICE

The CR11 Card Reader reads standard 80-column punched data cards.

#### 6.2 DRIVER FEATURES AND CAPABILITIES

The VAX/VMS card reader driver provides the following capabilities:

- Multiple controllers of the same type; for example, more than one CR11 can be used on the system
- Binary, packed Hollerith, and translated 026 or 029 read modes
- Unsolicited interrupt support for automatic card reader input spooling
- Special card punch combinations to indicate an end-of-file condition and to set the translation mode
- Error recovery

The following sections describe the read modes, special card punch combinations, and error recovery in greater detail.

##### 6.2.1 Read Modes

VAX/VMS provides two card reader device/function-dependent modifier bits for read data operations: read packed Hollerith (IO\$M\_PACKED) and read binary (IO\$M\_BINARY). If IO\$M\_PACKED is set, the data is packed and stored in sequential bytes of the input buffer. If IO\$M\_BINARY is set, the data is read and stored in sequential words of the input buffer. IO\$M\_BINARY takes precedence over IO\$M\_PACKED.

The read mode can also be set by a set translation mode card (see Section 6.2.2.2) or by the Set Mode function (see Section 6.4.3).

## CARD READER DRIVER

### 6.2.2 Special Card Punch Combinations

The VAX/VMS card reader driver recognizes three special card punch combinations in column 1 of a card. One combination signals an end-of-file condition. The other two combinations set the current translation mode.

**6.2.2.1 End-of-File Condition** - A card with the 12-11-0-1-6-7-8-9 holes punched in column 1 signals an end-of-file condition. If the read mode is binary, the first eight columns must contain this punch combination.

**6.2.2.2 Set Translation Mode** - If the read mode is nonbinary, nonpacked Hollerith (the IO\$M\_BINARY and IO\$M\_PACKED function modifiers are not set), the current translation mode can be set to the 026 or 029 punch code. A card with the 12-2-4-8 holes punched in column 1 sets the translation mode to the 026 code. A card with the 12-0-2-4-6-8 holes punched in column 1 sets the translation mode to the 029 code. The translation mode can be changed as often as required.

If a translation mode card contains punched information in columns 2 through 80, it is ignored.

Logical, virtual, and physical read functions result in only one card being read. If a translation mode card is read, the read function is not completed and another card is read immediately.

### 6.2.3 Error Recovery

The VAX/VMS card reader driver performs the following error recovery operations:

- If the card reader is off line for 30 seconds, a "device not ready" message is sent to the system operator.
- If a recoverable card reader failure is detected, a "device not ready" message is sent to the system operator every 30 seconds.
- The current operation is retried every two seconds to test for a changed situation, for example, the removal of an error condition.
- The current I/O operation can be canceled at the next timeout without the card reader being on line. When the card reader comes on line, device operation resumes automatically.

There are four categories of card reader failures:

- Pick check -- the next card cannot be delivered from the input hopper to the read mechanism.
- Stack check -- the card just read did not stack properly in the output hopper.

## CARD READER DRIVER

- Hopper check -- either the output hopper is full or the input hopper is empty.
- Read check -- the last card was read incorrectly due to torn edges or punches after column 80.

Manual intervention is required if any of these errors occur. The recovery is transparent to the user program issuing the I/O request.

When a recoverable card reader failure is detected, a "device not ready" message is displayed on the system operator console. When this message is received, the card reader indicator lights should be examined to determine the reason for the failure. The indicator lights and the respective recovery procedures are:

- Pick check -- the next card cannot be delivered to the read mechanism. Remove the next card to be read from the input hopper and smooth the leading edge, that is, the edge that will enter the read mechanism first. Replace the card in the input hopper and press the RESET button. Card reader operation will resume automatically. If a pick check error occurs again on the same card, remove the card from the input hopper and repunch it. Place the duplicate card in the input hopper and press the RESET button. If the problem persists, either an adjustment is required or nonstandard cards are in the input hopper.
- Stack check -- the card just read did not stack properly in the output hopper. Remove the last card read from the output hopper and examine the condition. If it is excessively worn or mutilated, repunch it. Place either the duplicate or the original card in the read station of the input hopper and press the RESET button. Card reader operation will resume automatically. If the stack check error reoccurs immediately, an adjustment is required.
- Hopper check -- either the input hopper is empty or the output hopper is full. Examine the input hopper and, if empty, either load the next deck of input cards or an end of file card. If the input hopper is not empty, remove the cards that have accumulated in the output hopper and press the RESET button. Card reader operation will resume automatically.
- Read check -- the last card was read incorrectly. Remove the last card from the output hopper and examine its condition. If it is excessively worn, mutilated, or contains punches before column 0 or after column 80, repunch the card correcting any incorrect punches. Place either the original or duplicate card in the read station of the input hopper and press the RESET button. Card reader operation will resume automatically. If the read check error reoccurs immediately, an adjustment is necessary.

### 6.3 DEVICE INFORMATION

Users can obtain information on card reader characteristics by using the \$GETCHN and \$GETDEV system services (see Section 1.10). The information is returned in a user-specified buffer shown in Figure 6-1. Only the first three longwords of the buffer are shown in Figure 6-1 (Figure 1-8 shows the entire buffer).

## CARD READER DRIVER

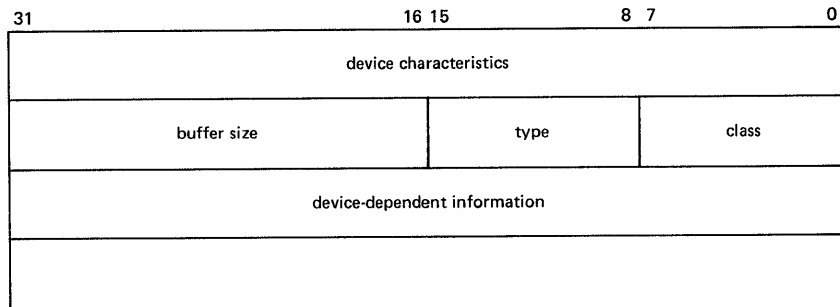


Figure 6-1 Card Reader Information

The device characteristics returned in the first longword are listed in Table 6-1.

Table 6-1  
Card Reader Device-Independent Characteristics

Dynamic Bit <sup>1</sup> (Conditionally Set)	Meaning
DEV\$M_AVL	Device is on line and available
Static Bits <sup>1</sup> (Always Set)	Meaning
DEV\$M_IDV	Device is capable of input
DEV\$M_REC	Device is record oriented

<sup>1</sup> Defined by the \$DEVDEF macro

The second longword contains information on device class and type, and the buffer size. The device class for card readers is DC\$\_CARD. The device type is DT\$\_CR11 for the CR11.

The \$DCDEF macro defines the device type and class names. The buffer size is the default to be used for all card reader devices (this default is 80 bytes).

The third longword contains device-dependent card reader characteristics. Table 6-2 lists these characteristics. The \$CRDEF macro defines the characteristics values.

CARD READER DRIVER

Table 6-2  
Device-Dependent Information for Card Readers

Value	Meaning
CR\$V_TMODE CR\$\$_TMODE	Specifies the translation mode for nonbinary, nonpacked Hollerith data transfers. <sup>1</sup> Possible values are:
CR\$K_T026	Translate according to 026 punch code
CR\$K_T029	Translate according to 029 punch code

<sup>1</sup> Section 6.2.2.2 describes the set translation mode punch code.

6.4 CARD READER FUNCTION CODES

The VAX/VMS card reader can perform logical, virtual, and physical I/O functions. Table 6-3 lists these functions and their function codes. These functions are described in more detail in the following paragraphs.

Table 6-3  
Card Reader I/O Functions

Function Code and Arguments	Type <sup>1</sup>	Function Modifiers	Function
IO\$_READBLK P1,P2	L	IO\$_BINARY IO\$_PACKED	Read logical block
IO\$_READVBLK P1,P2	V	IO\$_BINARY IO\$_PACKED	Read virtual block
IO\$_READPBLK P1,P2	P	IO\$_BINARY IO\$_PACKED	Read physical block
IO\$_SENSEMODE	L		Sense the card reader characteristics and return them in the I/O status block
IO\$_SETMODE P1	L		Set card reader characteristics for subsequent operations
IO\$_SETCHAR P1	P		Set card reader characteristics for subsequent operations

<sup>1</sup> V = virtual; L = logical; P = physical

## CARD READER DRIVER

### 6.4.1 Read

This function reads data from the next card in the card reader input hopper into the designated memory buffer in the specified format. Only one card is read each time a read function is specified.

VAX/VMS provides three read function codes:

- IO\$\_READVBLK - read virtual block
- IO\$\_READLBLK - read logical block
- IO\$\_READPBLK - read physical block

Two function-dependent arguments are used with these codes:

- P1 -- the starting virtual address of the buffer that is to receive the data
- P2 -- the number of bytes that are to be read in the specified format

The read binary function modifier (IO\$M\_BINARY) and the read packed Hollerith function modifier (IO\$M\_PACKED) can be used with all read functions. If IO\$M\_BINARY is specified, successive columns of data are stored in sequential word locations of the input buffer. If IO\$M\_PACKED is specified, successive columns of data are packed and stored in sequential byte locations of the input buffer. If neither of these function modifiers are specified, successive columns of data are translated in the current mode (026 or 029) and stored in sequential bytes of the input buffer. Figure 6-2 shows how data is stored by IO\$M\_BINARY and IO\$M\_PACKED.

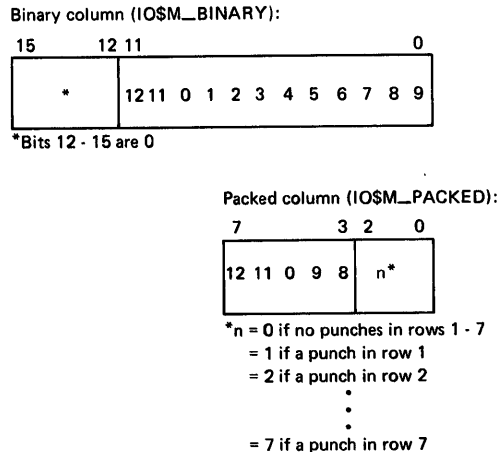


Figure 6-2 Binary and Packed Column Storage

Regardless of the byte count specified by the P2 argument, a maximum of 160 bytes of data for binary read operations and 80 bytes of data for nonbinary read operations (IO\$M\_PACKED, or 026 or 029 modes) are transferred to the input buffer. If P2 specifies less than the maximum quantity for the respective mode, only the number of bytes specified are transferred; any remaining buffer locations are not filled with data.

## CARD READER DRIVER

### 6.4.2 Sense Card Reader Mode

This function senses the current device-dependent card reader characteristics and returns them in the second longword of the I/O status block (see Table 6-5). No device/function dependent arguments are used with IO\$\_SENSEMODE.

### 6.4.3 Set Mode

Set mode operations affect the operation and characteristics of the associated card reader device. VAX/VMS defines two types of set mode functions:

- Set Mode
- Set Characteristic

6.4.3.1 **Set Mode** - The Set Mode function affects the characteristics of the associated card reader. Set Mode is a logical I/O function and requires the access privilege necessary to perform logical I/O. A single function code is provided:

- IO\$\_SETMODE

This function takes the following device/function dependent argument:

- P1 -- the address of a characteristics buffer

Figure 6-3 shows the quadword Set Mode characteristics buffer.

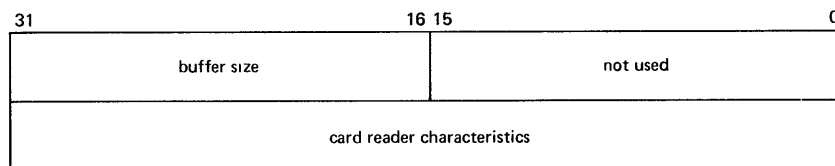


Figure 6-3 Set Mode Characteristics Buffer

Table 6-4 lists the card reader characteristics and their meanings. The \$CRDEF macro defines the characteristics values.

**CARD READER DRIVER**

Table 6-4  
Set Mode and Set Characteristic Card Reader Characteristics

Value 1	Meaning
CR\$V_TMODE CR\$\$_TMODE	Specifies the translation mode for nonbinary, nonpacked Hollerith data transfers. Possible values are:
CR\$K_T026	Translate according to 026 punch code
CR\$K_T029	Translate according to 029 punch code

\* If neither the 026 or 029 mode is specified, the default mode can be set by the SET CARD\_READER command.

6.4.3.2 Set Characteristic - The Set Characteristic function also affects the characteristics of the associated card reader device. Set Characteristic is a physical I/O function and requires the access privilege necessary to perform physical I/O functions. A single function code is provided:

- IO\$\_SETCHAR

This function takes the following device/function dependent argument:

- P1 -- the address of a characteristics buffer

Figure 6-4 shows the Set Characteristic characteristics buffer.

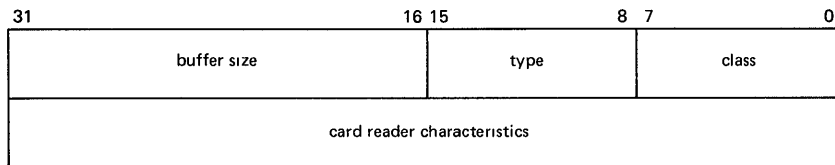


Figure 6-4 Set Characteristic Buffer

The device type value is DT\$\_CR11. The device class value is DC\$\_CARD. Table 6-4 lists the card reader characteristics for the Set Characteristic function.

### 6.5 I/O STATUS BLOCK

The I/O status block (IOSB) format for QIO functions on the card reader is shown in Figure 6-5. Table 6-5 lists the status returns for these functions. Table 6-2 lists the device-dependent data returned in the second longword. The IO\$\_SENSEMODE function can be used to obtain this data.

## CARD READER DRIVER

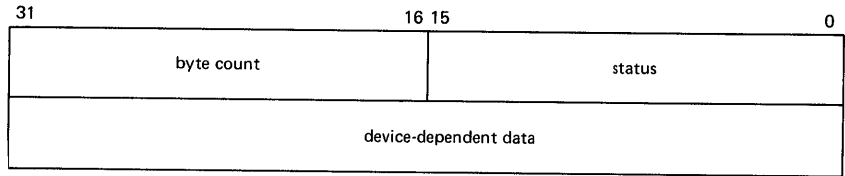


Figure 6-5 IOSB Contents

Table 6-5  
Status Returns for Card Reader

Status	Meaning
SS\$ _NORMAL	Successful completion of the operation specified in the QIO request. The second word of the IOSB can be examined to determine the actual number of bytes written to the buffer.
SS\$ _DATAOVERUN	Data overrun. Column data was delivered to the controller data buffer before previous data had been read by the driver.
SS\$ _ENDOFFILE	End-of-file condition. An end-of-file card was encountered during the read operation.



CHAPTER 7  
MAILBOX DRIVER

VAX/VMS supports a virtual device, called a mailbox, that is used for communication between processes. Mailboxes provide a controlled and synchronized method for processes to exchange data. Although mailboxes transfer information in much the same way that other I/O devices do, they are not actual devices. Rather, mailboxes are software implemented devices that can perform read and write operations.

7.1 MAILBOX OPERATIONS

Software mailboxes can be compared to the actual metal boxes used for mail delivery. As shown in Table 7-1, both types of mailbox perform similar operations.

Table 7-1  
Mailbox Read and Write Operations

Operation	Use of Conventional Mailboxes	Use of VAX/VMS Software Mailboxes
Receive Mail	Resident checks mailbox to see if any mail was delivered. If so, picks it up, opens it, and reads it.	A process initiates a read to a mailbox to obtain data sent by another process. The process reads data if a message was previously transmitted to the mailbox.
Receive Notification of Mail	The mail carrier leaves notification to the resident that mail can be picked up at the post office.	A process specifies that it wants to be notified through an AST when a message is sent to the mailbox.

(Continued on next page)

## MAILBOX DRIVER

Table 7-1 (Cont.)  
Mailbox Read and Write Operations

Operation	Use of Conventional Mailboxes	Use of VAX/VMS Software Mailboxes
Send Mail (without notification of receipt)	The resident leaves mail addressed to another person in the mailbox, but neither waits for nor expects notification of its delivery.	A process initiates a write request to a mailbox to transmit data to another process. The sending process does not wait until the data is read by the receiving process before completing the I/O operation.
Send Mail (with notification of receipt)	The resident leaves mail addressed to another person in the mailbox and asks to be notified of its delivery.	A process initiates a write request to a mailbox to transmit data to another process. The sending process waits until the receiving process reads the data before completing the I/O operation.
Reject Mail	The resident discards junk mail.	The receiving process rejects messages from the mailbox, sorts out unwanted messages, and responds only to useful messages.

### 7.1.1 Creating Mailboxes

A process uses the Create Mailbox and Assign Channel (\$CREMBX) system service to create a mailbox, and assign a channel and logical name to it. The system enters the logical name in either the system (permanent mailbox) or group (temporary mailbox) logical name table and gives it an equivalence name of MBn, where n is a unique unit number.

\$CREMBX also establishes the characteristics of the mailbox. These characteristics include a protection mask, permanence indicator, maximum message size, and buffer quota.

Other processes can assign additional channels to the mailbox using the Assign I/O Channel (\$ASSIGN) system service. The mailbox is identified by its logical name both when it is created and when it is assigned channels by cooperating processes.

Figure 7-1 illustrates the use of \$CREMBX and \$ASSIGN.

Creating mailboxes requires privilege. If sufficient dynamic memory for the mailbox data structure is not available, a resource wait may occur if resource wait mode is enabled.

The programming example at the end of this chapter (Section 7.5) illustrates mailbox creation and interprocess communication.

## MAILBOX DRIVER

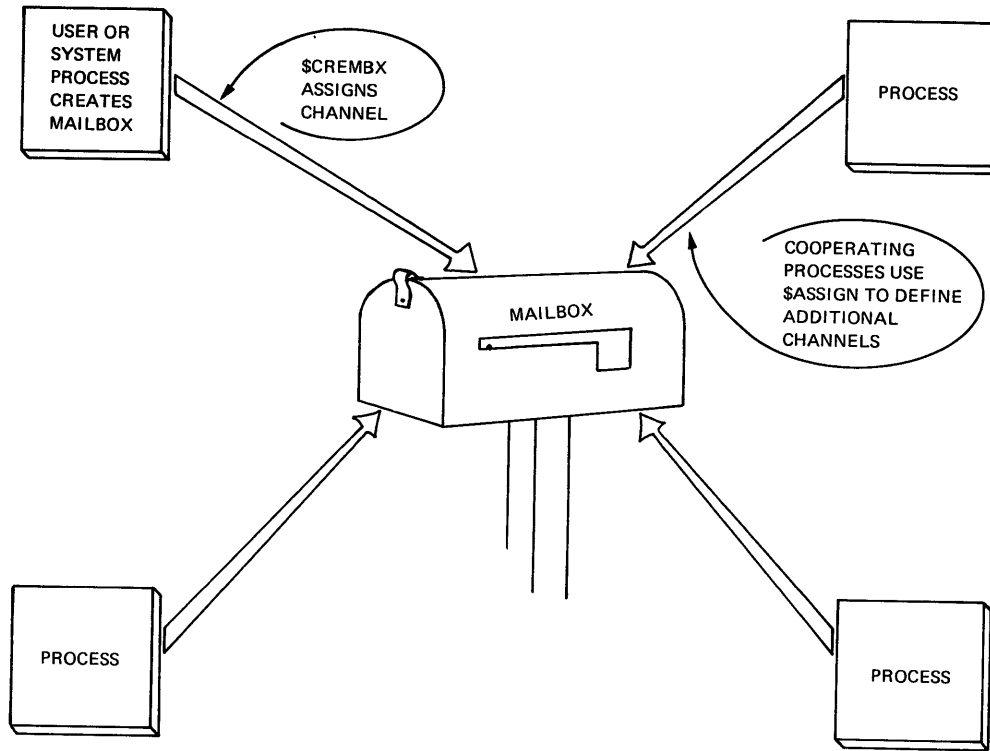


Figure 7-1 Multiple Mailbox Channels

### 7.1.2 Deleting Mailboxes

The system maintains a count of all channels assigned to a temporary mailbox. As each process finishes using a mailbox, it deassigns the channel using the Deassign I/O Channel (`$DASSGN`) system service. The channel count is decremented by one. The system automatically deletes the mailbox when no more channels are assigned to it (that is, when the channel count reaches 0).

Permanent mailboxes must be explicitly deleted using the Delete Mailbox (`$DELMBX`) system service. This can occur at any time. However, the mailbox is actually deleted when no processes have channels assigned to it.

When a mailbox is deleted, its message buffer quota is returned to the process that created it.

### 7.1.3 Mailbox Message Format

There is no standardized format for mailbox messages and none is imposed on users. Figure 7-2 shows a typical mailbox message format. Other types of messages can take different formats; for an example, see Figure 2-1 in Section 2.2.5.

## MAILBOX DRIVER

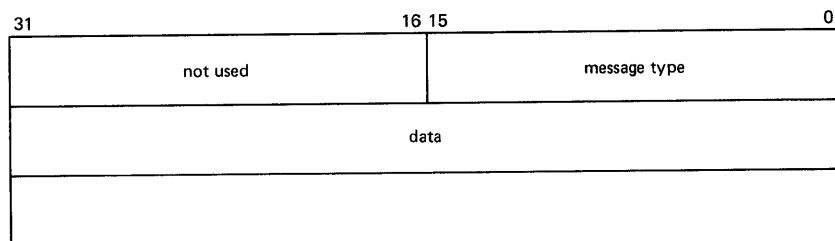


Figure 7-2 Typical Mailbox Message Format

### 7.2 DEVICE INFORMATION

Users can obtain information on mailbox characteristics by using the \$GETCHN and \$GETDEV system services (see Section 1.10). The information is returned in a user-specified buffer. The first three longwords of the buffer are shown in Figure 7-3 (Figure 1-8 shows the entire buffer).

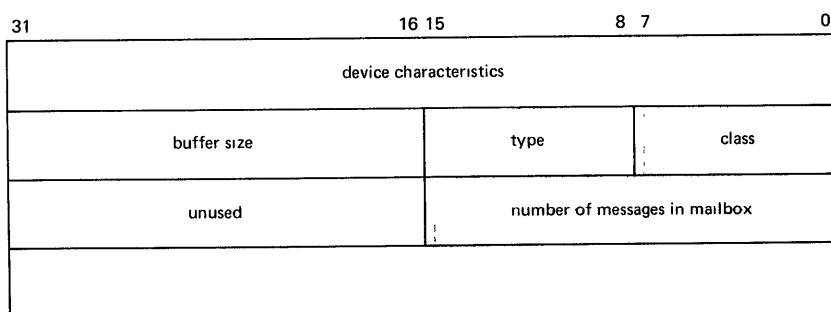


Figure 7-3 Mailbox Information

The first longword in the buffer contains the device characteristics values listed in Table 7-2. The \$DEVDEF macro defines these values.

Table 7-2  
Mailbox Characteristics

Dynamic Bit (Conditionally Set)	Meaning
DEV\$M_SHR	Shareable device
Static Bits (Always Set)	Meaning
DEV\$M_REC	Record-oriented device
DEV\$M_IDV	Device is capable of input
DEV\$M_ODV	Device is capable of output
DEV\$M_MBX	Mailbox device

## MAILBOX DRIVER

The second longword of the buffer contains information on the device class and type, and the buffer size. The device class is DC\$\_MBX. The device type is DT\$\_MBX. The \$DCDEF macro defines these symbols. The buffer size is the maximum message size in bytes.

### 7.3 MAILBOX FUNCTION CODES

The VAX/VMS mailbox I/O functions are: read, write, write end-of-file, and set attention AST.

No buffered I/O byte count quota checking is performed on mailbox I/O messages. Instead, the byte count or buffer quota of the mailbox is checked for sufficient space to buffer the message being sent. The buffered I/O quota and AST quota are also checked.

#### 7.3.1 Read

Read mailbox QIO requests are used to obtain messages written by other processes. The three mailbox functions and their codes are:

- IO\$\_READVBLK - read virtual block
- IO\$\_READLBLK - read logical block
- IO\$\_READPBLK - read physical block

These function codes take two device/function-dependent arguments:

- P1 -- the starting virtual address of the buffer that is to receive the message read
- P2 -- the size of the buffer in bytes (limited by the maximum message size for the mailbox)

One function modifier can be specified with a QIO read request:

- IO\$M\_NOW -- the I/O operation is completed immediately with no wait for a write request from another process

Figure 7-4 illustrates the read mailbox functions; in this figure, Process A reads a mailbox message written by Process B. As the figure indicates, a mailbox read request requires a corresponding mailbox write request (except in the case of an error). The requests can be made in any sequence; that is, the read request can either precede or follow the write request.

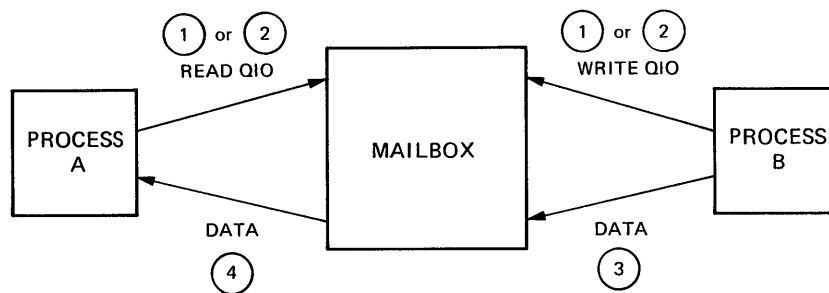
Two possibilities exist if Process A issues a read request before Process B issues a write request. If Process A did not specify the function modifier IO\$M\_NOW, Process A's request is queued during the wait for Process B to issue the write request. When this request occurs, the data is transferred from Process B, through the system buffers, to Process A to complete the I/O operation.

However, if Process A did specify the IO\$M\_NOW function modifier, the read operation is completed immediately. That is, Process A's request is not queued during the wait for the message from Process B, and no data is transferred from Process B to Process A.

## MAILBOX DRIVER

If Process B sends a message (with no function modifier; see Section 7.3.2) before Process A issues a read request (with or without a function modifier), Process A finds a waiting message in the mailbox. The data is transferred and the I/O operation is completed immediately.

To issue the read request, Process A can specify any of the read QIO function codes; all perform the same operation.



NOTE: Numbers indicate order of events.

Figure 7-4 Read Mailbox

### 7.3.2 Write

Write mailbox QIO requests are used to transfer data from a process to a mailbox. The three mailbox functions and their QIO function codes are:

- IO\$WRITEVBLK -- write virtual block
- IO\$WRITELBLK -- write logical block
- IO\$WRITEPBLK -- write physical block

These function codes take two device/function-dependent arguments:

- P1 -- the starting virtual address of the buffer that contains the message being written
- P2 -- the size of the buffer in bytes (limited by the maximum message size for the mailbox)

One function modifier can be specified with a QIO write request:

- IO\$M\_NOW - the I/O operation is completed immediately with no wait for another process to read the mailbox message

Figure 7-5 illustrates the write mailbox function; in this figure, Process A writes a message to be read by Process B. As in the read request example above, a mailbox write request requires a corresponding mailbox read request (unless an error occurs), and the requests can be made in any sequence.

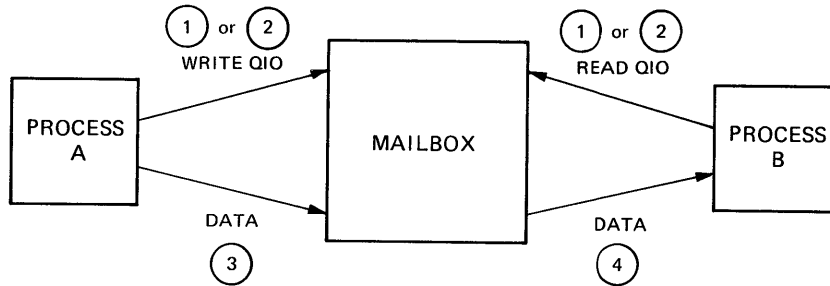
Two possibilities exist if Process A issues a write request before Process B issues a read request. If Process A did not specify the function modifier IO\$M\_NOW, Process A's write request is queued during the wait for Process B to issue a read request. When this request occurs, the data is transferred from Process A to Process B to complete the I/O operation.

## MAILBOX DRIVER

However, if Process A did specify the IO\$M NOW function modifier, the write operation is completed immediately. The data is available to Process B and is transferred when Process B issues a read request.

If Process B issues a read request (with no function modifier) before Process A issues a write request (with or without the function modifier), Process A finds a waiting request in the mailbox. The data is transferred and the I/O operation is completed immediately.

To issue the write request, Process A can specify any of the write QIO function codes; all perform the same operation.



NOTE: Numbers indicate order of events.

Figure 7-5 Write Mailbox

### 7.3.3 Write End-of-File Message

Write End-of-File Message QIO requests are used to insert a special message in the mailbox. The process that reads the end-of-file message is returned the status code SS\$\_ENDOFFILE in the I/O status block. No data is transferred. This function takes no arguments or function modifiers. VAX/VMS provides a single function code:

- IO\$\_WRITEOF -- write end-of-file message

### 7.3.4 Set Attention AST

Set Attention AST QIO requests are used to specify that an AST be given to notify the requesting process when a cooperating process places an unsolicited read or write request in a designated mailbox. Because the AST only occurs when the read or write request arrives from a cooperating process, the requesting process need not repeatedly check the mailbox status.

The Set Attention AST functions and their function codes are:

- IO\$\_SETMODE!IO\$\_READATTN - read attention AST
- IO\$\_SETMODE!IO\$\_WRTATTN - write attention AST

These function codes take two device/function-dependent arguments:

- P1 -- AST address (request notification is disabled if the address is 0)

## MAILBOX DRIVER

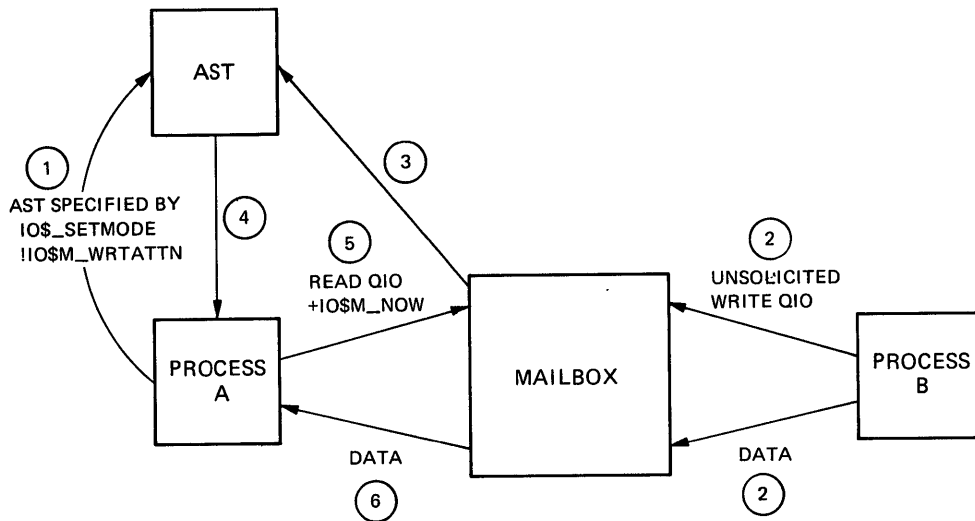
- P2 -- AST parameter returned in the argument list when the AST service routine is called
- P3 -- access mode to deliver AST; maximized with requester's mode

These functions are one-time AST enables; they must be explicitly re-enabled once the AST has been delivered if the user desires notification of the next unsolicited request. Both types of enables, and more than one of the each type, can be set at the same time. The number of enables is limited only by the AST quota for the process.

Figure 7-6 illustrates the write attention AST function. In this figure, an AST is set to notify Process A when Process B sends an unsolicited message.

Process A uses the `IO$_SETMODE!IO$_M_WRTATTN` function to request an AST. When Process B sends a message to the mailbox, the AST is delivered to Process A. Process A responds to the AST by issuing a read request to the mailbox. The function modifier `IO$_M_NOW` is included in the read request. The data is then transferred to complete the I/O operation.

If several requesting processes have set ASTs for unsolicited messages at the same mailbox, all ASTs are delivered when the first unsolicited message is placed in the mailbox. However, only the first process to respond to the AST with a read request receives the data. Thus, when the next process to respond to an AST issues a read request to the mailbox, it may find the mailbox empty. If this request does not include the function modifier `IO$_M_NOW`, it will be queued during the wait for the next message to arrive in the mailbox.



NOTE: Numbers indicate order of events.

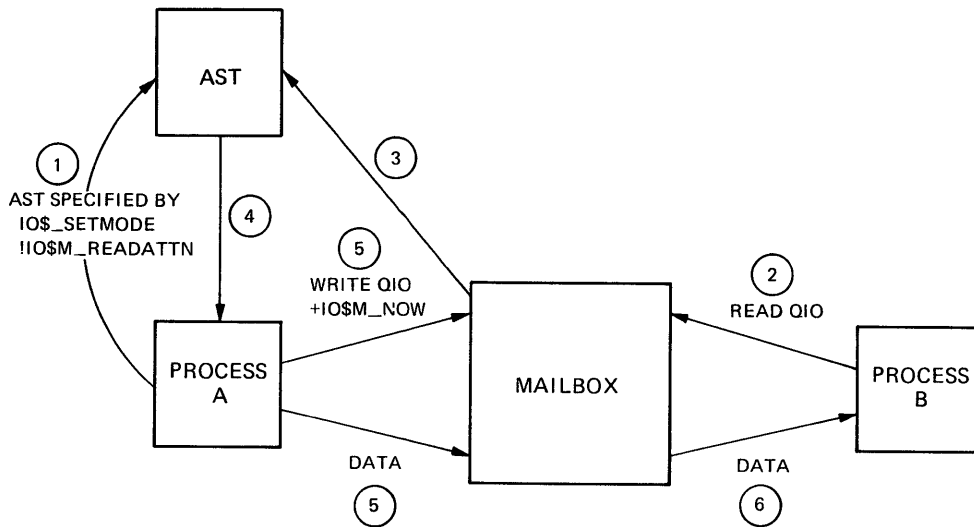
Figure 7-6 Write Attention AST (Read Unsolicited Data)

Figure 7-7 illustrates the read attention AST function. In this figure, an AST is set to notify Process A when Process B issues a read request for which no message is available.

## MAILBOX DRIVER

Process A uses the IO\$ SETMODE!IO\$M\_READATTN function to specify an AST. When Process B issues a read request to the mailbox, the AST is delivered to Process A. Process A responds to the AST by sending a message to the mailbox. The data is then transferred to complete the I/O operation.

If several requesting processes have set ASTs for read requests at the same mailbox, all ASTs are delivered when the first read request is placed in the mailbox. Only the first process to respond with a write request is able to transfer data to Process B.

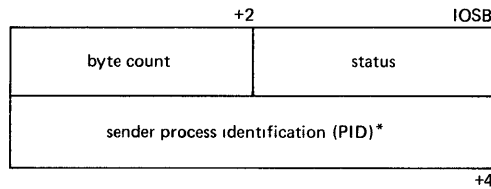


NOTE. Numbers indicate order of events.

Figure 7-7 Read Attention AST

### 7.4 I/O STATUS BLOCK

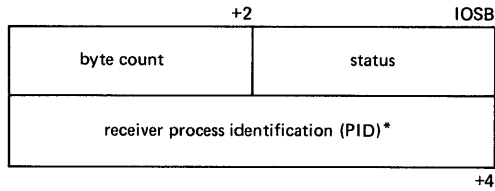
The I/O status blocks (IOSB) for mailbox read and write QIO functions are shown in Figures 7-8 and 7-9. Table 7-3 lists the status returns for these functions.



\*0 if the sender was a system process

Figure 7-8 IOSB Contents - Read Function

## MAILBOX DRIVER



\*0 if IO\$M\_NOW was specified

Figure 7-9 IOSB Contents - Write Function

Table 7-3  
Mailbox QIO Status Returns

Status	Meaning
SS\$_NORMAL	Successful completion. The operation specified in the QIO was completed successfully. The second word of the IOSB can be examined to determine the number of bytes transferred.
SS\$_ENDOFFILE	No message available at the mailbox or end-of-file (IO\$_ENDOFFILE) message read.

### 7.5 PROGRAMMING EXAMPLE

The following program creates a mailbox and puts some mail in it; no matching read is pending on the mailbox. First, the program illustrates that if the function modifier IO\$M NOW is not used when mail is deposited, the write function will wait until a read operation is performed on the mailbox. In this case, IO\$M NOW is specified and the program continues after the mail is left in the mailbox.

Next, the mailbox is read. If there was no mail in the mailbox the program would wait at this point because IO\$M NOW is not specified. IO\$M NOW should be specified if there is any doubt concerning the availability of data in the mailbox and it is important for the program not to wait.

It is up to the user to coordinate what data goes into and out of mailboxes. In this example the process reads its own message. Normally, two mailboxes are used for interprocess communication: one for sending data from process A to process B, and one for sending data from process B to process A. If a program is arranged in this manner, there is no possibility of a process reading its own message.

## MAILBOX DRIVER

### MAILBOX DRIVER PROGRAMMING EXAMPLE /01/

```

;
; Define necessary symbols
;
        SIODEF                                ;Define I/O function codes
;
; Allocate storage for necessary data structures
;
; Allocate terminal device name string and descriptor
;
DEVICE_DESCR:
        .LONG    20$-10$                      ;Length of name string
        .LONG    10$                          ;Address of name string
10$:    .ASCII   /TERMINAL/                   ;Name string of output device
20$:    ;Reference label
;
; Allocate space to store assigned channel number
;
DEVICE_CHANNEL:
        .BLKW   1                              ;Channel number
;
; Allocate mailbox name string and descriptor
;
MAILBOX_NAME:
        .LONG    ENDBOX-NAMEBOX               ;Length of name string
        .LONG    NAMEBOX                      ;Address of name string
NAMEBOX: .ASCII   /146_MAIN_ST/              ;Name string
;
ENDBOX: ;Reference label
;
; Allocate space to store assigned channel number
;
MAILBOX_CHANNEL:
        .BLKW   1                              ;Channel number
;
; Now allocate space to store the outgoing and incoming messages
;
IN_BOX_BUFFER:
        .BLKB   40                            ;Allocate 40 bytes for received message
        IN_LENGTH=-IN_BOX_BUFFER             ;Define input buffer length
;
OUT_BOX_BUFFER:
        .ASCII   /SHEEP ARE VERY DIM/        ;Message to send
        OUT_LENGTH=-OUT_BOX_BUFFER           ;Define length of message to send
;
; Now allocate space for the I/O status quadword
;
STATUS: .QUAD   1                              ;I/O status quadword
;
; Now the program. A mailbox is created and a channel is assigned
; to the terminal. A message is put in the mailbox and a message
; is received from the mailbox (the same message).The contents of
; the mailbox are then printed on the terminal.
;
START:  .WORD    0                              ;Entry mask
        $CREMBX_S CHAN=MAILBOX_CHANNEL,-;Channel is the mailbox
        PROMSK=#^X0000,-                      ;No protection
        BUFOUD=#^X0060,-                      ;Buffer quota is hex 60
        LOGNAM=MAILBOX_NAME,-                 ;Logical name descriptor
        MAXMSG=#^X0060                         ;Maximum message is hex 60
        CMPW    #SSS_NORMAL,RO                ;Test for normal return
        BSBW    ERROR_CHECK                   ;See if all well
        $ASSIGN_S
        DEVNAM=DEVICE_DESCR                   ;Assign channel
        CHAN=DEVICE_CHANNEL                   ;Device descriptor
        CMPW    #SSS_NORMAL,RO                ;Channel
        BSBW    ERROR_CHECK                   ;Test for normal return
        ;See if all is well
;
; Now we will write the message to the mailbox using the function
; modifier IOSM_NOW so that we may continue without waiting for a
; read on the mailbox
;
        SIOIOW_S FUNC=#IOS_WRITEVBLK!IOSM_NOW,- ;write message NOW
        CHAN=MAILBOX_CHANNEL,-                ;To the mailbox channel
        P1=OUT_BOX_BUFFER,-                   ;Buffer to write

```

# MAILBOX DRIVER

```

        CMPW    P2=#OUT_LENGTH      ;How much to write
        BSBW    #SS$_NORMAL,R0     ;Test for normal return
        BSBW    ERROR_CHECK        ;See if all is well

;
; Now the mailbox is read
;
        SQIOW_S  FUNC=#IOS_READVBLK,- ;read box
        CHAN=MAILBOX_CHANNEL,-      ;Mailbox channel
        IOSB=STATUS,-               ;Define status to receive message length
        P1=IN_BOX_BUFFER,-         ;Where to read it
        P2=#IN_LENGTH              ;How much
        CMPW    #SS$_NORMAL,R0     ;Test for normal return
        BSBW    ERROR_CHECK        ;See if all is well

;
; Now we find out how much mail was in the box and print it to the terminal
; The amount of mail read is held in STATUS+2
;
        MOVZWL  STATUS+2,R2         ;Put byte count into R2
        SQIOW_S  FUNC=#IOS_WRITEVBLK,- ;Function is write
        CHAN=DEVICE_CHANNEL,-      ;To the terminal
        P1=IN_BOX_BUFFER,-         ;Address of buffer to write
        P2=R2,-                    ;How much to write
        P4=#32                     ;Carriage control (1H ,)

;
; we now deassign the channel and exit
;
EXIT:   SDASSGN_S  CHAN=DEVICE_CHANNEL ;Deassign channel
        RET                               ;Return

;
; This is the error checking part of the program. Normally some kind of
; error recovery would be attempted here but not for this example.
;
ERROR_CHECK:
        BNEO    EXIT                ;Directive failed so exit
        RSB                               ;Else return

        .END      START

```

## CHAPTER 8

### DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER

This chapter describes the use of the VAX/VMS DMC11 Synchronous Communications Line Interface driver. The DMC11 provides a direct-memory-access interface (DMA) between two computer systems using the DIGITAL Data Communications Message Protocol (see Section 8.1.1 below). The DMC11 supports DMA data transfers of up to 16K bytes at rates of up to 1 million baud for local operation (over coaxial cable) and 56,000 baud for remote operation (using modems). Both full- and half-duplex modes are supported.

The DMC11 is a message-oriented communications line interface that is used primarily to link two separate but cooperating computer systems.

#### 8.1 SUPPORTED DMC11 SYNCHRONOUS LINE INTERFACES

Table 8-1 lists the DMC11 options supported by VAX/VMS.

Table 8-1  
Supported DMC11 Options

Type	Use
DMC11-AR with DMC11-FA DMC11-AR with DMC11-DA	Remote DMC11 and EIA or V35/DDS line unit
DMC11-AL with DMC11-MD DMC11-AL with DMC11-MA	Local DMC11 and 1M bps or 56 bps

##### 8.1.1 DIGITAL Data Communications Message Protocol

To ensure reliable data transmission, the DIGITAL Data Communications Message Protocol (DDCMP) has been implemented, using a high-speed microprocessor, on the VAX-11/780 processor. For remote operations, a DMC11 can communicate with a different type of synchronous interface (or even a different type of computer), provided the remote system has implemented the DDCMP, version 4.

## DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER

The DDCMP detects errors on the communication line interconnecting the systems using a 16-bit Cyclic Redundancy Check (CRC). Errors are corrected, when necessary, by automatic message retransmission. Sequence numbers in message headers ensure that messages are delivered in the proper order with no omissions or duplications.

The DDCMP specification (Order No. AA-D599A-TC) provides more detailed information on the DDCMP.

### 8.2 DRIVER FEATURES AND CAPABILITIES

DMC11 driver capabilities include:

- A nonprivileged QIO interface to the DMC11. This allows use of the DMC11 as a raw-data channel.
- Unit attention conditions transmitted through attention ASTs and mailbox messages.
- Both full- and half-duplex operation.
- Interface design common to all communications devices supported by VAX/VMS.
- Error logging of all DMC11 microprocessor and line unit errors.
- On -line diagnostics.
- Separate transmit and receive quotas.

The following sections describe mailbox usage and I/O quotas.

#### 8.2.1 Mailbox Usage

The device owner process can associate a mailbox with a DMC11 by using the \$ASSIGN system service (see Section 7.1.2). The mailbox is used to receive messages that signal attention conditions about the unit. As illustrated in Figure 8-1, these messages have the following context and format:

- Message type; this can be any one of the following:

Message type	Meaning
MSG\$_XM_DATAVL	Data is available
MSG\$_XM_SHUTDN	Unit has been shutdown
MSG\$_XM_ATT	A disconnect, timeout, or data check occurred

The \$MSGDEF macro is used to define message types

- Physical unit number of the DMC11
- Size (count) of the ASCII device name string
- Device name string

## DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER

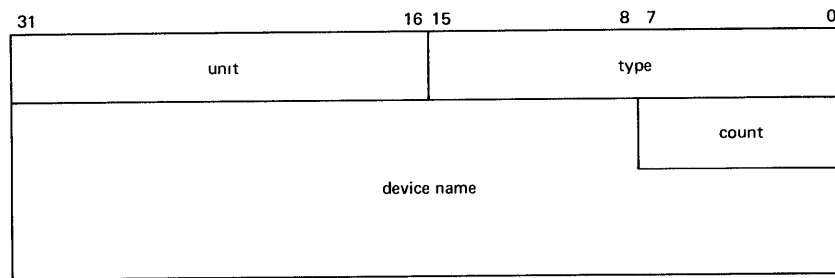


Figure 8-1 Mailbox Message Format

### 8.2.2 Quotas

Transmit operations are considered direct I/O operations and are limited by the process's direct I/O quota.

The quotas for the receive buffer free list (see Section 8.4.3.4) are the process's buffered I/O count and buffered I/O byte limit. After start up, the transient byte count and the buffered I/O byte limit are adjusted.

### 8.2.3 Power Failure

When a system power failure occurs, no DMC11 recovery is possible. The device is in a fatal error state and is shutdown.

## 8.3 DEVICE INFORMATION

Users can obtain information on device characteristics by using the \$GETCHN and \$GETDEV system services (see Section 1.10). The information is returned in a user-specified buffer shown in Figure 8-2. Only the first three longwords of the buffer are shown in Figure 8-2 (Figure 1-8 shows the entire buffer).

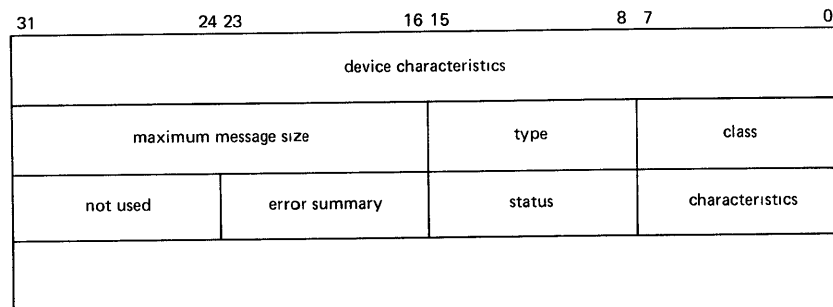


Figure 8-2 DMC11 Information

The first longword in the buffer contains the device characteristics values listed in Table 8-2. The \$DEVDEF macro defines these values.

DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER

Table 8-2  
DMC11 Device Characteristics

Dynamic bit (Conditionally Set)	Meaning
DEV\$M_NET	Network device
Static bits (Always Set)	Meaning
DEV\$M_ODV DEV\$M_IDV	Output device Input device

The second longword contains information on the device class and type, and the maximum message size. The device class for the DMC11 is DC\$\_SCOM. Table 8-3 lists the device types. The device class and types are defined by the \$DCDEF macro.

Table 8-3  
DMC11 Device Types

Device Type	Meaning <sup>1</sup>
DT\$_XM_ARDA	DMC11-AR with DMC11-DA
DT\$_XM_ARFA	DMC11-AR with DMC11-FA
DT\$_XM_ALMD	DMC11-AL with DMC11-MD
DT\$_XM_ALMA	DMC11-AL with DMC11-MA

<sup>1</sup> Table 8-1 describes the different device types

The maximum message size is the maximum send or receive message size for the unit. Messages greater than 512 bytes on modem controlled lines are more prone to transmission errors and therefore may require more retransmissions.

The third longword contains unit characteristics and status, and an error summary.

Unit characteristics bits govern the DDCMP operating mode. They are defined by the \$XMDEF macro and can be read or set. Table 8-4 lists the unit characteristics values and their meanings.

DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER

Table 8-4  
DMC11 Unit Characteristics

Characteristic	Meaning <sup>1</sup>
XM\$M_CHR_MOP	DDCMP maintenance mode
XM\$M_CHR_SLAVE	DDCMP half-duplex slave station
XM\$M_CHR_HDPLX	DDCMP half-duplex
XM\$M_CHR_LOOPB	DDCMP loop back
XM\$M_CHR_INHER	Inhibit error logging. Since DMC11 operation is independent of individual I/O operations, error logging cannot be inhibited on a per-request basis.
XM\$M_CHR_MBX	Shows the status of the mailbox that can be associated with the unit; if this bit is set, the mailbox is enabled to receive messages signaling unsolicited data. (This bit can also be changed as a subfunction of read or write QIO functions)

<sup>1</sup> Section 8.1.1 describes the DDCMP

The status bits show the status of the unit and the line. The values are defined by the \$XMDEF macro. They can be read, set, or cleared as indicated. Table 8-5 lists the status values and their meanings.

Table 8-5  
DMC11 Unit and Line Status

Status	Meaning
XM\$M_STS_ACTIVE	Protocol is active. This bit is set when IO\$ SETMODE!IO\$ STARTUP is done and cleared when the unit is shut down. (Read only.)
XM\$M_STS_TIMO	Timeout. If set, indicates that the receiving computer is unresponsive. DDCMP time outs. (Read or clear.)
XM\$M_STS_ORUN	Data overrun. If set, indicates that a message was received but lost due to the lack of a receive buffer. (Read or clear.)

(Continued on next page)

## DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER

Table 8-5 (Cont.)  
DMC11 Unit and Line Status

Status	Meaning
XM\$M_STS_DCHK	Data check. If set, indicates that a retransmission threshold has been exceeded. (Read or clear.)
XM\$M_STS_DISC	If set, indicates that the Data Set Ready (DSR) modem line went from on to off. (Read or clear.)

The error summary bits are set only when the driver must shut down the DMC11 because a fatal error occurred. These are read-only bits that are cleared by any of the IO\$ SETMODE functions (see Section 8.4.3). The XM\$M\_STS\_ACTIVE status bit is clear if any error summary bit is set. Table 8-6 lists the error summary bit values and their meanings.

Table 8-6  
Error Summary Bits

Error Summary Bit	Meaning
XM\$M_ERR_MAINT	DDCMP maintenance message received
XM\$M_ERR_START	DDCMP START message received
XM\$M_ERR_LOST	Data was lost when a message was received that was longer than the specified maximum message size.
XM\$M_ERR_FATAL	An unexpected hardware/software error occurred.

### 8.4 DMC11 FUNCTION CODES

The basic DMC11 function codes are read, write, and set mode. All three functions take function modifiers.

#### 8.4.1 Read

VAX/VMS provides three read function codes:

- IO\$ READLBLK - read logical block
- IO\$ READPBLK - read physical block
- IO\$ READVBLK - read virtual block

## DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER

Received messages are multi-buffered in system dynamic memory and then copied to the user's address space when the read operation is performed.

The QIO arguments for the three function codes are:

- P1 -- the starting virtual address of the buffer that is to receive data
- P2 -- the size of the receive buffer in bytes

The read QIO functions can take two function modifiers:

- IO\$M\_DSABLMBX - disables use of the associated mailbox for unsolicited data notification
- IO\$M\_NOW - complete the read operation immediately if no message is available

### 8.4.2 Write

VAX/VMS provides three write QIO function codes:

- IO\$\_WRITEBLK - write logical block
- IO\$\_WRITEPBLK - write physical block
- IO\$\_WRITEVBLK - write virtual block

Transmitted messages are sent directly from the requesting process's buffer.

The QIO arguments for the three function codes are:

- P1 -- the starting virtual address of the buffer containing the data to be transmitted
- P2 -- the size of the buffer in bytes

The message size specified by P2 cannot be larger than the maximum send message size for the unit (see Section 8.3). If a message larger than the maximum size is sent, a status of SS\$\_DATAOVERUN is returned in the I/O status block.

The write QIO functions can take one function modifier:

- IO\$M\_ENABLMBX - enable use of the associated mailbox

### 8.4.3 Set Mode

Set mode operations are used to perform protocol, operational, and program/driver interface operations with the DMC11. VAX/VMS defines five types of set mode functions:

- Set Mode
- Set Characteristics
- Enable Attention AST

**DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER**

- Set Mode and Shut Down Unit
- Set Mode and Start Unit

8.4.3.1 **Set Mode and Set Characteristics** - These functions set device characteristics such as maximum message size. VAX/VMS provides two function codes:

- IO\$\_SETMODE - set mode (requires logical I/O privilege)
- IO\$\_SETCHAR - set characteristics (requires physical I/O privilege)

One argument is used with these function codes:

- P1 -- the virtual address of the quadword characteristics buffer block if the characteristics are to be set. If this argument is zero, only the unit status and characteristics are returned in the I/O status block (see Section 8.5). Figure 8-3 shows the P1 characteristics block.

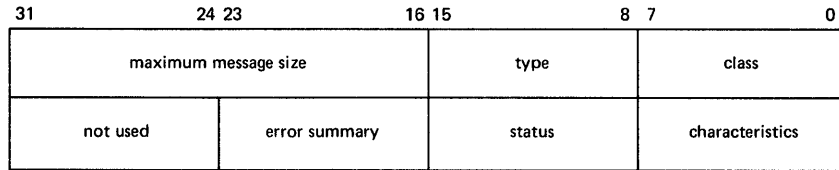


Figure 8-3 P1 Characteristics Block

In the buffer designated by P1 the device class is DC\$\_SCOM. Table 8-3 (in Section 8.3) lists the device types. The maximum message size describes the maximum send or receive message size.

The second longword contains device/function dependent characteristics: unit characteristics, status, and error summary bits. Any of the characteristics values and some of the status values can be set or cleared (see Tables 8-4, 8-5, and 8-6).

If the unit is active (XM\$\_STS\_ACTIVE is set), the action of a Set Mode or Set Characteristics function with a characteristics buffer is to clear the status bits or the error summary bits. If the unit is not active, the status bits or the error summary bits can be cleared, and the maximum message size, type, device class, and unit characteristics can be changed.

8.4.3.2 **Enable Attention AST** - This function enables an AST to be queued when an attention condition occurs on the unit. An AST is queued when the driver sets or clears either an error summary bit or any of the unit status bits, or when a message is available and there is no waiting read request. The Enable Attention AST function is legal at any time, regardless of the condition of the unit status bits.

## DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER

VAX/VMS provides two function codes:

- IO\$\_SETMODE!IO\$\_ATTNAST - enable attention AST
- IO\$\_SETCHAR!IO\$\_ATTNAST - enable attention AST

Enable Attention AST is a single (one-time) enable. After the AST occurs, it must be explicitly re-enabled by the function before the AST can occur again. The function code is also used to disable the AST. The function is subject to AST quotas.

The Enable Attention AST functions take the following device/function dependent arguments:

- P1 -- address of AST service routine or 0 for disable
- P2 -- (ignored)
- P3 -- access mode to deliver AST

The AST service routine is called with an argument list. The first argument is the current value of the device/function dependent characteristics longword shown in Figure 8-3. The access mode specified by P3 is maximized with the requester's access mode.

**8.4.3.3 Set Mode and Shut Down Unit** - This function stops the operation on an active unit (XM\$\_STS\_ACTIVE must be set) and then resets the unit characteristics.

VAX/VMS provides two function codes:

- IO\$\_SETMODE!IO\$\_SHUTDOWN - shut down unit
- IO\$\_SETCHAR!IO\$\_SHUTDOWN - shut down unit

These functions take one device/function dependent argument:

- P1 -- the virtual address of the quadword characteristics block (Figure 8-3) if modes are to be set after shutdown. P1 is 0 if modes are not to be set after shutdown.

These functions stop the DMC11 microprocessor and release all outstanding message blocks; any messages that have not been read are lost. The characteristics are reset after shutdown. Except for the signaling of attention ASTs and mailbox messages, the action of these functions is the same as the action of the driver when shutdown occurs because of a fatal error.

**8.4.3.4 Set Mode and Start Unit** - This function sets the characteristics and starts the protocol on the associated unit. VAX/VMS provides two function codes:

- IO\$\_SETMODE!IO\$\_STARTUP - start unit
- IO\$\_SETCHAR!IO\$\_STARTUP - start unit

## DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER

These functions take the following device/function dependent arguments:

- P1 -- the virtual the address of the quadword characteristics block (Figure 8-3) if the characteristics are to be set. Characteristics are set before the device is started.
- P2 -- (ignored)
- P3 -- the number of pre-allocated receive-message blocks to ensure the availability of buffers to receive messages.

The total quota taken from the process's buffered I/O byte count quota is the DMC11 work space, plus the number of receive-message buffers specified by P3 times the maximum message size. For example, if six 200-byte, buffers are required, the total quota taken is 1456 bytes:

```
    256 (DMC11 work space)
+ 1200 (number of buffers X buffer size)
-----
    1456 (total quota taken)
```

This quota is returned to the process when shutdown occurs.

Receive-message blocks are used by the driver to receive messages that arrive independent of QIO read request timing. When a message arrives, it is matched with any outstanding read requests. If there are no outstanding read requests, the message is queued and an attention AST or mailbox message is generated. (IO\$SETMODE!IO\$M\_ATTNAST or IO\$SETCHAR!IO\$M\_ATTNAST must be set to enable an attention AST; IO\$M\_ENABLMBX must be used to enable a mailbox message.)

When read, the receive-message block is returned to the receive-message "free list" defined by P3. If the "free list" is empty, no receive-messages are possible. In this case, a data lost condition can be generated if a message arrives. This nonfatal condition is reported by device-dependent data and an attention AST.

### 8.5 I/O STATUS BLOCK

The I/O status block (IOSB) usage for all DMC11 QIO functions is shown in Figure 8-4. Table 8-7 lists the status returns for these functions.

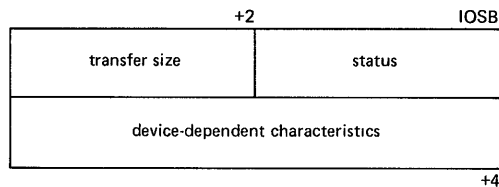


Figure 8-4 IOSB Content

In Figure 8-4, the transfer size at IOSB+2 is the actual number of bytes transferred. Table 8-4 lists the device-dependent characteristics returned at IOSB+4. These characteristics can also be obtained by using the \$GETCHN and \$GETDEV system services (see Section 8.3).

**DMC11 SYNCHRONOUS COMMUNICATIONS LINE INTERFACE DRIVER**

Table 8-7  
Status Returns for DMC11

Status	Meaning
SS\$_ABORT	Fatal hardware error or I/O canceled in progress.
SS\$_DATAOVERUN	Message received overran buffer allocated (read), or message too big (write).
SS\$_ENDOFFILE	No data available (read) when IO\$_NOW was specified.
SS\$_NORMAL	Operation was successfully completed (read, write, or set modes).
SS\$_DEVOFFLINE	Device protocol not started (read or write). The function is inconsistent with the current state of the unit (Set Mode).
SS\$_DEVACTIVE	The function is inconsistent with the current state of the unit.



## CHAPTER 9

### QIO INTERFACE TO FILE SYSTEM ACPs

An ancillary control process (ACP) is a process that interfaces between the user process and the driver, and performs functions that supplement the driver's functions. Virtual I/O operations involving file-structured devices (disks and magnetic tapes) often require ACP intervention. In most cases, ACP intervention is requested by VAX-11 Record Management Services (RMS) and is transparent to the user process. However, user processes can request ACP functions directly by issuing a QIO request and specifying an ACP function code, as shown in Figure 9-1.

The DECnet/VAX User's Guide describes network ACP (NETACP) interface operations.

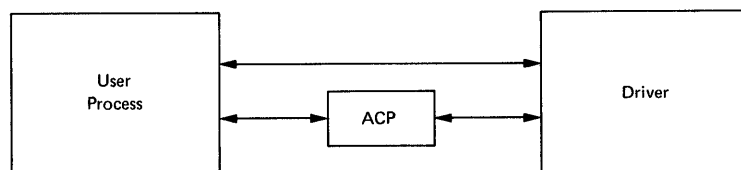


Figure 9-1 ACP QIO Interface

This chapter describes the QIO interface to ACPs for disk and magnetic tape devices (file system ACPs). The example program in Chapter 4 performs QIO operations to the magnetic tape ACP.

#### 9.1 ACP FUNCTIONS AND ENCODING

All VAX/VMS ACP functions can be expressed using six function codes and five function modifiers. The function codes are:

- IO\$\_CREATE -- creates a directory entry or file.
- IO\$\_ACCESS -- searches a directory for a specified file and accesses that file, if it is found.
- \$IO\_DEACCESS -- deaccesses a file and, if specified, writes the final attributes in the file header.
- IO\$\_MODIFY -- modifies the file attributes and/or file allocation.
- IO\$\_DELETE -- deletes a directory entry and/or file header.

## QIO INTERFACE TO FILE SYSTEM ACPS

- IO\$\_MOUNT -- mounts a volume; requires mount privilege.
- IO\$\_ACPCONTROL -- performs miscellaneous control functions.

Appendix A describes the function codes in more detail. The function modifiers are:

- IO\$\_ACCESS -- opens the file on the user's channel; applies only to the create and access functions
- IO\$\_CREATE -- creates a file; applies only to the create and access functions
- IO\$\_DELETE -- deletes the file (or marks it for deletion); applicable only to create and delete functions to disk devices.
- IO\$\_DMOUNT -- dismounts a volume; applies only to the ACP control function

### 9.1.1 ACP Device/Function-Dependent Arguments

In addition to the function codes and modifiers, VAX/VMS ACPS take five device/function-dependent arguments, as shown in Figure 9-2.

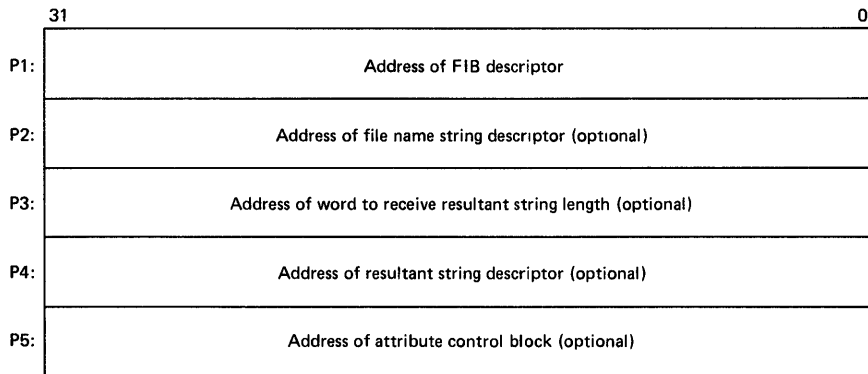


Figure 9-2 ACP Device/Function-Dependent Arguments

The first argument, P1, is the address of the File Information Block descriptor. Section 9.2 describes the FIB in detail.

The second argument, P2, is an optional argument used in directory operations. It specifies the address of the descriptor for the file name string to be entered in the directory. The file name itself must be in read/write memory.

Argument P3 is the address of a word to receive the resultant file name string length. The resultant string is not padded. The actual length is returned in P3. P4 is the address of a descriptor for a buffer to receive the resultant file name string. Both these arguments are optional.

The fifth argument, P5, is an optional argument containing the address of the attribute control block. Section 9.3 describes the attribute control block in detail.

## QIO INTERFACE TO FILE SYSTEM ACPS

Figure 9-3 shows the format for the descriptors.

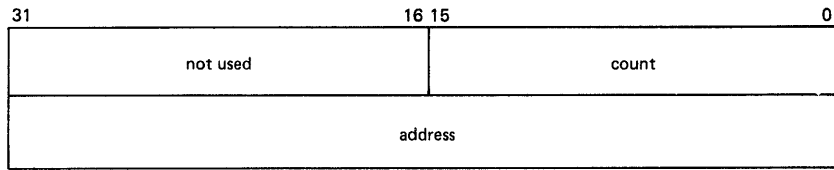
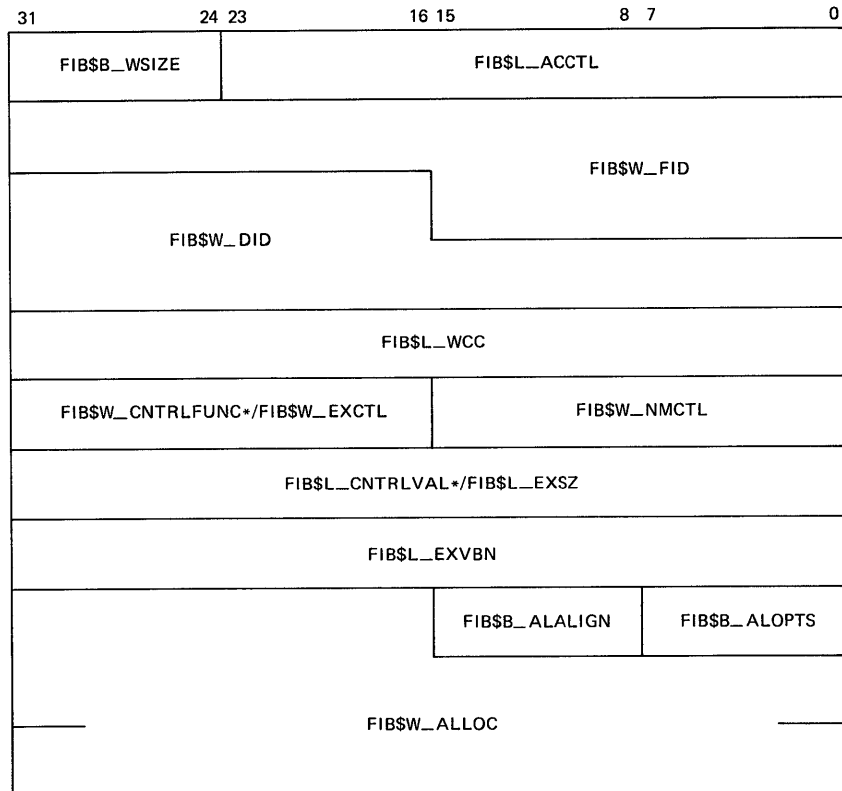


Figure 9-3 ACP Device/Function Argument Descriptor Format

### 9.2 FILE INFORMATION BLOCK

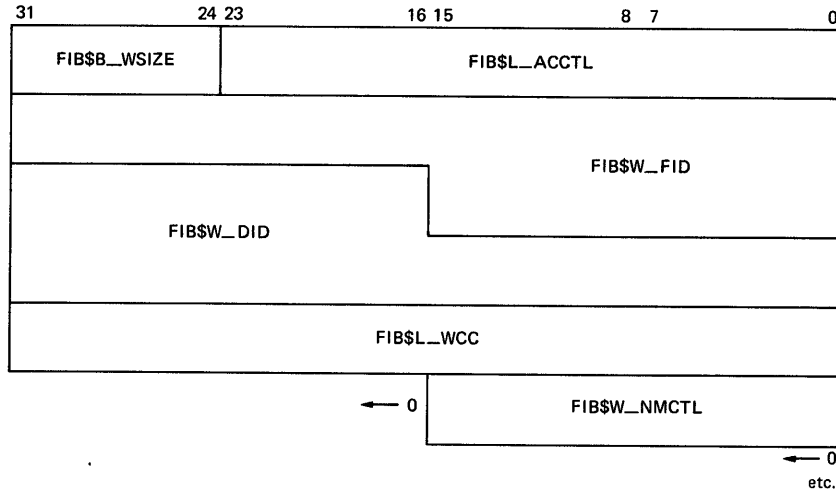
The File Information Block (FIB) contains much of the information that is exchanged between the user process and an ACP. Figure 9-4 shows the format of the FIB. Because the FIB is passed by a descriptor (Pl in Figure 9-2), its length can vary. Thus, a short FIB can be used in ACP calls that do not need arguments toward the end of the FIB. The ACP automatically zero-extends a short FIB. Figure 9-5 shows the format of a typical short FIB, in this case one that would be used to open an existing file. Table 9-1 lists the values of the FIB fields.



\*Only for magnetic tape

Figure 9-4 File Information Block Format

**QIO INTERFACE TO FILE SYSTEM ACPS**



**Figure 9-5 Typical Short File Information Block**

**QIO INTERFACE TO FILE SYSTEM ACPS**

Table 9-1  
Contents of the File Information Block

Field	Field Bits	Meaning
FIB\$ <u>L</u> _ACCTL		Specifies field values that control access to the file. The following bits are defined:
	FIB\$ <u>M</u> _WRITE	Set for write access; clear for read-only access.
	FIB\$ <u>M</u> _NOREAD	Set to deny read access to others
	FIB\$ <u>M</u> _NOWRITE	Set to deny write access to others
	FIB\$ <u>M</u> _DLOCK	Set to enable deaccess lock (close check). Only for disk devices.
		Used to flag a file as inconsistent in the event the program currently modifying the file terminates abnormally. If the program then closes the file without performing a write attributes operation, the file is marked as locked and cannot be accessed until it is unlocked.
	FIB\$ <u>M</u> _SEQONLY	Set for sequential-only access. Only for disk devices.
	FIB\$ <u>M</u> _REWIND	Set to rewind magnetic tape before access
	FIB\$ <u>M</u> _CURPOS	Set to create magnetic tape file at current position (note: a magnetic tape file will be created at the end of the volume set if neither FIB\$ <u>M</u> _REWIND nor FIB\$ <u>M</u> _CURPOS are set). If the tape is not positioned at the end of a file, FIB\$ <u>M</u> _CURPOS creates the file at the next file position.
	FIB\$ <u>M</u> _UPDATE	Set to position at start of a magnetic tape file when opening file for write; clear to position at end-of-file
	FIB\$ <u>M</u> _READCK	Set to enable read checking of the file
	FIB\$ <u>M</u> _WRITECK	Set to enable write checking of the file

(Continued on next page)

QIO INTERFACE TO FILE SYSTEM ACPS

Table 9-1 (Cont.)  
Contents of the File Information Block

Field	Field Bits	Meaning
FIB\$B_WSIZE		Controls the size of the file window used to map a disk file. The ACP will use the volume default if FIB\$B_WSIZE is 0. A value of 1 to 127 indicates the number of retrieval pointers to be allocated to the window. A value of -1 indicates that the window should be as large as necessary to map the entire file.
FIB\$W_FID		Specifies the file identification. The user supplies the file identifier when it is known; the ACP returns the file identifier when it becomes known, for example, as a result of a create or directory lookup
FIB\$W_DID		Contains the file identifier of the directory file
FIB\$L_WCC		Maintains position context when processing wild card directory operations
FIB\$W_NMCTL		Controls the processing of a name string in a directory operation. The following bits are defined:
	FIB\$M_WILD	Set if name string contains wild cards
	FIB\$M_ALLNAM	Set to match all name field values
	FIB\$M_ALLTYP	Set to match all field type values
	FIB\$M_ALLVER	Set to match all version field values
	FIB\$M_NEWVER	Set to create file of same name with next higher version number. Only for disk devices.
	FIB\$M_SUPERSEDE	Set to supersede an existing file of the same name, type, and version. Only for disk devices.
	FIB\$M_FINDFID	Set to search a directory for the file identifier in FIB\$W_FID
	FIB\$M_LOWVER	Set on return from a CREATE if a lower numbered version of the file exists. Only for disk devices.

(Continued on next page)

QIO INTERFACE TO FILE SYSTEM ACPS

Table 9-1 (Cont.)  
Contents of the File Information Block

Field	Field Bits	Meaning
FIB\$W_NMCTL (Cont.)	FIB\$M_HIGHVER	Set on return from a CREATE if a higher numbered version of the file exists. Only for disk devices.
FIB\$W_EXCTL		Specifies extend control for disk devices. The following bits are defined:
	FIB\$M_EXTEND <sup>1</sup>	Set to enable extension
	FIB\$M_TRUNC <sup>1</sup>	Set to enable truncation
	FIB\$M_NOHDREXT	Set to inhibit generation of extension file headers
	FIB\$M_ALCON	Allocate contiguous space
	FIB\$M_ALCONB	Allocate contiguous space, best effort
	FIB\$M_FILCON	Mark file contiguous
	FIB\$M_ALDEF	Allocate the extend size (FIB\$L_EXSZ) or the system default, whichever is greater
FIB\$W_CNTRLFUNC		Controls magnetic tape functions. In an ACPCONTROL function, the FIB\$W_CNTRLFUNC field can contain one of the following values:
	FIB\$C_REWINDFIL	Rewind to beginning of file
	FIB\$C_POSEND	Position to end of volume set
	FIB\$C_NEXTVOL	Force next volume
	FIB\$C_SPACE	Space n blocks forward or in reverse
	FIB\$C_REWINDVOL	Rewind to beginning of volume set
FIB\$L_EXSZ		Specifies the number of blocks to allocate to, or remove from, a disk file depending on the FIB\$W_EXCTL field configuration. For truncate operations, this field must contain 0.

<sup>1</sup> Only one of these can be set at one time; that is, extension cannot be enabled at the same time truncation is enabled, and vice versa.

(Continued on next page)

QIO INTERFACE TO FILE SYSTEM ACPS

Table 9-1 (Cont.)  
Contents of the File Information Block

Field	Field Bits	Meaning
FIB\$ <u>L</u> _EXSZ (Cont.)		The number of blocks actually allocated or removed is returned in this longword. The value may differ from the user-requested value because of adjustments for cluster boundaries. More blocks are allocated and fewer blocks removed to meet cluster boundaries.
FIB\$ <u>L</u> _CNTRLVAL		If FIB\$C SPACE is indicated, the FIB\$ <u>L</u> _CNTRLVAL field specifies the number of magnetic tape blocks to space forward if positive or space backward if negative.
FIB\$ <u>L</u> _EXVBN		Specifies the starting disk file virtual block number at which the allocated blocks are to appear in an extend operation, or the first virtual block number to be removed in a truncate operation. For extend operations, this field must contain either the end-of-file block number plus 1, or 0. For truncate operations, this field specifies the first virtual block number to be removed. The actual starting virtual block number of the extend or truncate operation is returned in this field.

Table 9-2 shows which FIB fields and field values are used in the respective ACP QIO functions. Some of the FIB fields and values are applicable only to disk devices or only to magnetic tape devices. See Table 9-1.

QIO INTERFACE TO FILE SYSTEM ACPS

Table 9-2  
FIB Argument Usage in ACP QIO Functions

I/O Function	Application Arguments	
	FIB Field	Field Values
IO\$_CREATE	FIB\$L_ACCTL	FIB\$M_WRITE FIB\$M_NOREAD FIB\$M_NOWRITE FIB\$M_DLOCK FIB\$M_SEQONLY FIB\$M_REWIND FIB\$M_CURPOS FIB\$M_UPDATE FIB\$M_READCK FIB\$M_WRITECK
	FIB\$B_WSIZE	
	FIB\$W_FID <sup>2</sup>	
	FIB\$W_DID	
	FIB\$W_NMCTL	FIB\$M_NEWVER FIB\$M_SUPERSEDE FIB\$M_LOWER <sup>5</sup> FIB\$M_HIGHER <sup>5</sup>
	FIB\$W_EXCTL	FIB\$M_EXTEND <sup>6</sup> FIB\$M_TRUNC FIB\$M_NOHDREXT FIB\$M_ALCON FIB\$M_ALCONB FIB\$M_FILCON FIB\$M_ALDDEF
IO\$_ACCESS	FIB\$L_EXSZ	
	FIB\$L_ACCTL	FIB\$M_WRITE FIB\$M_NOREAD FIB\$M_NOWRITE FIB\$M_DLOCK FIB\$M_SEQONLY FIB\$M_REWIND FIB\$M_CURPOS FIB\$M_UPDATE FIB\$M_READCK FIB\$M_WRITECK
	FIB\$B_WSIZE FIB\$W_FID <sup>3</sup>	

(Continued on next page)

QIO INTERFACE TO FILE SYSTEM ACPS

Table 9-2 (Cont.)  
FIB Argument Usage in ACP QIO Functions

I/O Function	Application Arguments	
	FIB Field	Field Values
IO\$ ACCESS (CONT.)	FIB\$W_DID	
	FIB\$L_WCC <sup>4</sup>	
	FIB\$W_NMCTL	FIB\$M_WILD FIB\$M_ALLNAM FIB\$M_ALLTYPE FIB\$M_ALLVER
IO\$ DEACCESS (no arguments used)	FIB\$W_FID <sup>3</sup>	
IO\$ MODIFY	FIB\$W_DID	
	FIB\$L_WCC <sup>4</sup>	
	FIB\$W_NMCTL	FIB\$M_WILD FIB\$M_ALLNAM FIB\$M_ALLTYPE FIB\$M_ALLVER
	FIB\$W_EXCTL	FIB\$M_EXTEND*** FIB\$M_TRUNC FIB\$M_NOHDREXT FIB\$M_ALCON FIB\$M_ALCONB FIB\$M_FILCON FIB\$M_ALDEF
	FIB\$L_EXSZ	
	FIB\$L_EXVBN	
IO\$ DELETE	FIB\$W_FID <sup>3</sup>	
	FIB\$W_DID	
	FIB\$L_WCC <sup>4</sup>	
	FIB\$W_NMCTL	FIB\$M_WILD FIB\$M_ALLNAM FIB\$M_ALLTYP FIB\$M_ALLVER

(Continued on next page)

**QIO INTERFACE TO FILE SYSTEM ACPS**

Table 9-2 (Cont.)  
FIB Argument Usage in ACP QIO Functions

I/O Function	Application Arguments	
IO\$ MOUNT (no arguments used) IO\$_ACPCONTROL <sup>1</sup>	FIB Field  FIB\$W_CNTRLFUNC  FIB\$L_CNTRLVAL (if FIB\$C_SPACE is indicated)	Field Values  FIB\$C_REWINDFIL FIB\$C_POSEND FIB\$C_NEXTVOL FIB\$C_SPACE FIB\$C_REWINDVOL

1 If IO\$M\_DMOUNT is not set, the magnetic tape control function field specifies one of the field values listed.

2 If FIB\$W\_DID = 0 and IO\$M\_CREATE is not set; FIB\$W\_FID is an output argument if IO\$M\_CREATE is set.

3 If FIB\$W\_DID is 0; FIB\$W\_FID is an output argument if FIB\$W\_DID is not 0.

4 If FIB\$V\_WILD is set.

5 Output argument

6 Only FIB\$M\_EXTEND or FIB\$M\_TRUNC can be set at any given time; they cannot both be set at the same time.

### 9.3 ATTRIBUTE CONTROL BLOCK

The attribute control block contains the codes that control the reading and writing of file attributes, for example, file protection and record attributes. Device/function-dependent argument P5 specifies the address of this list. The list consists of a variable number of 2-longword control blocks, terminated by a zero longword, as shown in Figure 9-6. The maximum number of attribute control blocks in one list is 14. Table 9-3 describes the attribute control block fields.

QIO INTERFACE TO FILE SYSTEM ACPS

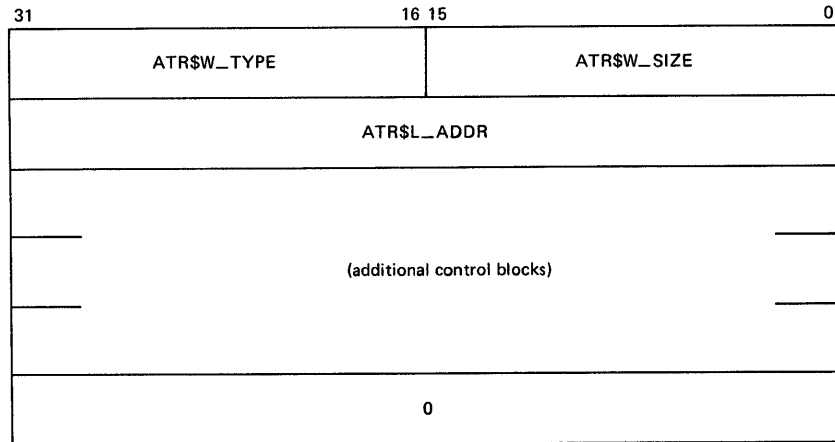


Figure 9-6 Attribute Control Block Format

Table 9-3  
Attribute Control Block Fields

Field	Meaning																
ATR\$W_SIZE	Specifies the number of bytes of the attribute to be transferred. Legal values are from 0 to the maximum size of the particular attribute (see Table 9-4).																
ATR\$W_TYPE	Identifies the individual attribute to be read or written.																
ATR\$L_ADDR	Contains the buffer address of the user's memory space to or from which the attribute is to be transferred. The particular I/O function determines whether the attribute is read or written, as follows:																
	<table border="0"> <thead> <tr> <th style="text-align: center;"><u>I/O Function</u></th> <th style="text-align: center;"><u>Operation</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">Create</td> <td style="text-align: center;">Write</td> </tr> <tr> <td style="text-align: center;">Access</td> <td style="text-align: center;">Read</td> </tr> <tr> <td style="text-align: center;">Deaccess</td> <td style="text-align: center;">Write</td> </tr> <tr> <td style="text-align: center;">Modify</td> <td style="text-align: center;">Write</td> </tr> <tr> <td style="text-align: center;">Delete</td> <td style="text-align: center;">Not used</td> </tr> <tr> <td style="text-align: center;">Mount</td> <td style="text-align: center;">Not used</td> </tr> <tr> <td style="text-align: center;">ACP Control</td> <td style="text-align: center;">Not used</td> </tr> </tbody> </table>	<u>I/O Function</u>	<u>Operation</u>	Create	Write	Access	Read	Deaccess	Write	Modify	Write	Delete	Not used	Mount	Not used	ACP Control	Not used
<u>I/O Function</u>	<u>Operation</u>																
Create	Write																
Access	Read																
Deaccess	Write																
Modify	Write																
Delete	Not used																
Mount	Not used																
ACP Control	Not used																

Table 9-4 lists the valid attributes for ACP QIO functions. The maximum size (in bytes) is determined by the required attribute configuration. For example, the file name uses only 6 bytes, but is always accompanied by the file type and file version - so a total of 10 bytes is required. Each attribute has two names: one for the code (for example, ATR\$C\_UCHAR); and one for the size (for example, ATR\$S\_UCHAR).

QIO INTERFACE TO FILE SYSTEM ACPS

Table 9-4  
ACP QIO Attributes

Attribute Name	Maximum Size (bytes)	Meaning
(unnamed)	5	Two-byte file owner UIC plus the next attribute and the first byte of ATR\$C_UCHAR. Used for compatibility mode only.
(unnamed)	3	Two-byte file protection plus the first byte of ATR\$C_UCHAR. Used for compatibility mode only.
ATR\$C_UCHAR ATR\$S_UCHAR	4	Four-byte file characteristics.
ATR\$C_RECATTR ATR\$S_RECATTR	32	Record attribute area.
ATR\$C_FILNAM ATR\$S_FILNAM	10	Six-byte Radix-50 file name plus ATR\$C_FILTYP and ATR\$C_FILVER.
ATR\$C_FILTYP ATR\$S_FILTYP	4	Two-byte Radix-50 file type plus ATR\$C_FILVER.
ATR\$C_FILVER ATR\$S_FILVER	2	Two-byte binary version number.
ATR\$C_EXPDAT ATR\$S_EXPDAT	7	Expiration date in ASCII.
ATR\$C_STATBLK ATR\$S_STATBLK	10	Statistics block.
ATR\$C_HEADER ATR\$S_HEADER	512	Complete file header.
ATR\$C_BLOCKSIZE ATR\$S_BLOCKSIZE	2	Magnetic tape block size.
ATR\$C_ASCDATES ATR\$S_ASCDATES	35	Revision count; revision date, creation date, and expiration date, in ASCII.
ATR\$C_ALCONTROL ATR\$S_ALCONTROL	14	Compatibility mode allocation data.
ATR\$C_ASCNAME ATR\$S_ASCNAME	20	File name, type, and version, in ASCII.
ATR\$C_CREDATE ATR\$S_CREDATE	8	64-bit creation date and time.
ATR\$C_REVDATE ATR\$S_REVDATE	8	64-bit revision date and time.
ATR\$C_EXPDATE ATR\$S_EXPDATE	8	64-bit expiration date and time.

(Continued on next page)

**QIO INTERFACE TO FILE SYSTEM ACPS**

Table 9-4 (Cont.)  
ACP QIO Attributes

Attribute Name	Maximum Size (bytes)	Meaning
ATR\$C_BAKDATE ATR\$S_BAKDATE	8	64-bit backup date and time.
ATR\$C_UIC ATR\$S_UIC	4	4-byte file owner UIC.
ATR\$C_FPRO ATR\$S_FPRO	2	File protection.
ATR\$C_RPRO ATR\$S_RPRO	2	Record protection.
ATR\$C_ACLEVEL ATR\$S_ACLEVEL	7	File access level.
ATR\$C_SEMASK ATR\$S_SEMASK	4	File security mask and limit.
ATR\$C_UIC_RO	4	4-byte file owner UIC (read only; ignored if write)

**9.4 I/O STATUS BLOCK**

Figure 9-7 shows the I/O status block (IOSB) for ACP QIO functions. Table 9-5 lists the status returns for these functions.

The file ACP returns a completion status code in the first longword of the IOSB. In an extend operation, the second longword is used to return the number of blocks allocated to the file. If a contiguous extend operation (FIB\$V\_ALCON) fails, the second longword is used to return the size of the file after truncation.

Values returned in the IOSB are most useful during operations in compatibility mode. When executing programs in the native mode, the user should use the values returned in the FIB locations.

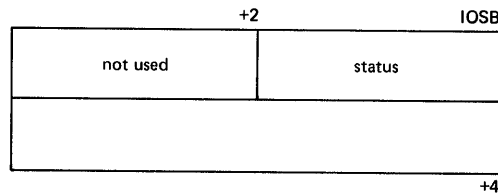


Figure 9-7 IOSB Contents - ACP QIO Functions

If an extend operation (including CREATE) was performed, IOSB+4 contains the number of blocks allocated, or the largest available contiguous space if a contiguous extend operation failed. If a truncate operation was performed, IOSB+4 contains the number of blocks added to the file size to reach the next cluster boundary.

QIO INTERFACE TO FILE SYSTEM ACPS

Table 9-5  
ACP QIO Status Returns

Status	Meaning
SS\$_ACCONFLICT	Access mode conflict. Requested access mode conflicted with existing file accesses, for example, an attempt to open a file for a write when the file is write locked.
SS\$_ACPVAFUL	The magnetic tape ACP's virtual address space is full. Since each volume set has a virtual page assigned to it, additional volume sets cannot be handled. Corrective action consists of starting a different ACP using the unique switch and MOUNT.
SS\$_BADATTRIB	Invalid attribute code or size specified in read or write attribute list.
SS\$_BADCHKSUM	Invalid checksum in the file header.
SS\$_BADFILEHDR	Invalid file header, for example, structure is inconsistent or the storage map indicates free blocks.
SS\$_BADFILENAME	Invalid syntax in file name string. The string contains illegal characters, or is larger than 9 characters.
SS\$_BADFILEVER	Invalid file version number, that is, a number greater than 32767.
SS\$_BADIRECTORY	Invalid directory file. The file is not a directory or the file contains invalid data.
SS\$_BADPARAM	Invalid parameter list. For example, the FIB contains options not applicable to this function.
SS\$_BLOCKCNTERR	Block count error. The number of blocks read differs from the number of blocks recorded in the trailer labels. There is a possibility that a record was skipped or an extra noise record was read.
SS\$_CREATED	File created by an ACCESS function with a CREATE function modifier.
SS\$_DEVICEFULL	Device full. No free blocks are available on the device or the number of contiguous blocks specified in a contiguous request is not available.
SS\$_DIRFULL	Directory is full. An error occurred while creating a disk file because the directory specified is full and cannot catalog any more entries. A directory is limited to 1024 blocks.

(Continued on next page)

**QIO INTERFACE TO FILE SYSTEM ACPS**

Table 9-5 (Cont.)  
ACP QIO Status Returns

Status	Meaning
SS\$_DUPFILENAME	Duplicate file name. Another directory entry with the same name, type, and version already exists.
SS\$_ENDOFFILE	End-of-file. End of allocated space encountered in a virtual I/O operation or an attempted truncation.
SS\$_FCPREADERR	FCP read error. An I/O error occurred when file structure data, for example, a directory, was read.
SS\$_FCPREWINDERR	File process rewind error. An I/O error occurred when rewinding a volume.
SS\$_FCSPACERR	File process space error. An I/O error occurred when spacing within a file or spacing files.
SS\$_FCPWITERR	FCP write error. An I/O error occurred when file structure data, for example, a directory, was written.
SS\$_FILELOCKED	File deaccess locked. Attempted to access a locked file. A file becomes locked when it is accessed with FIB\$_DLOCK set and then deaccessed without writing attributes.
SS\$_FILENUMCHK	File identifier number check. The index file contains invalid data.
SS\$_FILESEQCHK	File identifier sequence check. A directory entry points to a file that has been deleted.
SS\$_FILESTRUCT	Unsupported file structure. The file structure on the accessed volume is not compatible with the ACP. For example, an attempt was made to use a structure level 2 ACP with a structure level 1 disk.
SS\$_FILNOTEXP	File not expired. A magnetic tape file that has not expired cannot be written over unless the override expiration qualifier was specified in MOUNT.
SS\$_HEADERFULL	File header map area is full and header extension is inhibited. This can occur on a volume's index file in a CREATE operation.
SS\$_IDXFILEFULL	Volume index file is full. The maximum number of files specified at initialization time has been reached.
SS\$_ILLCNTRFUNC	Illegal control function. An illegal function is specified for IO\$_ACPCONTROL.

(Continued on next page)

**QIO INTERFACE TO FILE SYSTEM ACPS**

Table 9-5 (Cont.)  
ACP QIO Status Returns

Status	Meaning
SS\$_NOMOREFILES	No more files exist which match the given wild card in a file specification string. At least one file was found, that is, one match was made.
SS\$_NOPRIV	No privilege. Volume or file protection will not allow the requested operation.
SS\$_NOSUCHFILE	No such file. No file with the given file name or file identifier exists. Can be caused by a directory entry that points to a file that has been deleted.
SS\$_NOTAPEOP	No tape operator. There is no tape operator and a need to communicate with one exists, for example, the next volume in a volume set must be mounted.
SS\$_NOTLABELMT	Magnetic tape not labeled. A request to read a magnetic tape failed because the tape does not have standard labels.
SS\$_SUPERSEDE	An existing file of the same name, type, and version has been deleted by a CREATE function.
SS\$_TAPEPOSLOST	Magnetic tape position lost.
SS\$_TOOMANYVER	Too many versions. The maximum number of file versions already exists. All are higher versions than the one being created.
SS\$_WRTLCK	The device is software write locked or the hardware write lock switch on the drive is set.



## APPENDIX A

### DISK, MAGNETIC TAPE, AND ACP QIO FUNCTIONS

This appendix provides detailed descriptions of the common functions performed by the disk and magnetic tape drivers and the ACP QIO interface.

#### A.1 CREATE FILE

This virtual I/O function creates a directory entry and/or a file on a disk device, or a file on a magnetic tape device.

The function code is:

- IO\$CREATE

The function modifiers are:

- IO\$M\_CREATE
- IO\$M\_ACCESS
- IO\$M\_DELETE

The device/function-dependent arguments for IO\$CREATE are:

- P1 -- the address of the File Information Block (FIB) descriptor.
- P2 -- the address of the file name string descriptor (optional). If specified for a disk file, the name is entered in the directory specified by the FIB. If specified for a magnetic tape file, the name is the name of the created file.
- P3 -- the address of the word that is to receive the length of the resultant file name string (optional).
- P4 -- the address of a descriptor for a buffer that is to receive the resultant file name string (optional).
- P5 -- the address of a list of attribute descriptors (optional). If specified for a disk file, the indicated attributes are written to the file header. If specified for a magnetic tape file, P5 is the address of the descriptor list for the new file.

The name string is entered in the disk directory specified by the FIB\$W\_DID field of the FIB. If the resultant string descriptor is present, a string representing the full directory entry is returned. If the address of a word to receive the resultant string size is

## DISK, MAGNETIC TAPE, AND ACP QIO FUNCTIONS

specified, the size, in bytes, of the string is returned. A disk file can also be extended if `FIB$M_EXTEND` is set. The number of blocks allocated is returned in the second longword of the `IOSB`.

A disk file header is created if `IO$M_CREATE` is specified. (The file ID is returned in `FIB$W_FID`.) If an attribute list is present, the indicated attributes are written to the file header. If `IO$M_DELETE` is specified, the disk file is marked for deletion. This function modifier may only be used in conjunction with `IO$M_CREATE` and `IO$M_ACCESS`.

If `IO$M_ACCESS` is specified, the disk or magnetic tape file is accessed, that is, opened on the user's channel.

In the name control field (`FIB$W_NMCTL`) of the `FIB`, the `FIB$M_NEWVER` and `FIB$M_SUPERSEDE` bits function as described in Table 9-1; other flags are ignored. The wild card context field, `FIB$L_WCC`, is also ignored.

The `FIB$L_ACCTL` and `FIB$W_EXCTL` `FIB` fields are interpreted as described in Table 9-1.

Table A-1 lists the arguments for `IO$M_CREATE` in the order in which they are used. All other arguments are illegal and must be zero.

Table A-1  
`IO$M_CREATE` Arguments

<code>IO\$M_CREATE</code>
<code>FIB\$M_EXTEND</code> (disk only)
<code>FIB\$W_EXCTL</code> (disk only)
<code>FIB\$W_DID</code>
<code>FIB\$W_NMCTL</code>
<code>FIB\$L_WCC</code>
Attribute List
<code>IO\$M_ACCESS</code>
<code>FIB\$L_ACCTL</code>
<code>IO\$M_DELETE</code> (disk only)

### A.2 ACCESS FILE

This virtual I/O function searches a directory on a disk device, or a magnetic tape, for a specified file and accesses that file if found.

The function code is:

- `IO$M_ACCESS`

## DISK, MAGNETIC TAPE, AND ACP QIO FUNCTIONS

The function modifiers are:

- IO\$M\_CREATE
- IO\$M\_ACCESS

IO\$M\_CREATE changes the IO\$ ACCESS function code to IO\$ CREATE if the directory search failed with a "file not found" condition. The function is then re-executed as a CREATE. If IO\$M\_ACCESS is specified, the file is accessed. A file must be accessed before it can be read or written.

The device/function-dependent arguments for IO\$ ACCESS are:

- P1 -- the address of the File Information Block (FIB) descriptor.
- P2 -- the address of the file name string descriptor (optional). If specified for disks, the name is entered in the directory specified by the FIB. If specified for magnetic tapes, the name identifies the file being sought.
- P3 -- the address of the word that is to receive the length of the resultant file name string (optional).
- P4 -- the address of a descriptor for a buffer that is to receive the resultant file name string (optional).
- P5 -- the address of a list of attribute descriptors (optional). If specified for disks, the indicated attributes are written to the file header. If specified for magnetic tapes, the file attributes are returned to the user.

Initially, a search is made for the name string indicated in a directory specified in FIB\$W\_DID. If the resultant string descriptor is present, a string representing the full directory entry is returned. The size of the string is returned if the address of the resultant string size word is present. The file identifier is returned in FIB\$W\_FID.

Several other FIB fields are used in IO\$ ACCESS execution. In the FIB\$W\_NMCTL field, FIB\$M\_ALLNAM, FIB\$M\_ALLTYP, and FIB\$M\_ALLVER control matching of the name fields. If FIB\$M\_WILD is set, FIB\$W\_WCC indicates the position in the directory to resume the search; on return, this field contains the position of the directory entry found. The FIB\$L\_ACCTL field is interpreted as described in Table 9-1.

If an attribute list is present, the indicated file attributes are read.

Table A-2 lists the arguments for IO\$ ACCESS in the order in which they are used. All other arguments are illegal and must be 0.

## DISK, MAGNETIC TAPE, AND ACP QIO FUNCTIONS

Table A-2  
IO\$\_ACCESS Arguments

FIB\$W_DID
FIB\$W_NMCTL
FIB\$W_WCC
IO\$M_CREATE
IO\$M_ACCESS
FIB\$L_ACCTL
Attribute List
(Extend control data is ignored)

### A.3 DEACCESS FILE

This virtual I/O function deaccesses a file and, if specified, writes final attributes in the file header.

Attributes are written to a disk file if they are present and if the file was accessed for a write operation. (If a write operation and no attributes are specified, and if FIB\$M\_DLOCK is set, the file is checked to determine if it is locked, that is, access is inhibited.)

The function code is:

- IO\$\_DEACCESS

The device/function-dependent arguments for IO\$\_DEACCESS are:

- P1 -- the address of the File Information Block (FIB) descriptor.
- P2 -- the address of the file name string descriptor (optional). If specified, the name is entered in the directory specified by the FIB.
- P3 -- the address of the word that is to receive the length of the resultant file name string (optional).
- P4 -- the address of a descriptor for a buffer that is to receive the resultant file name string (optional).
- P5 -- the address of a list of attribute descriptors (optional). If specified, the indicated attributes are written to the file header.

Only two legal arguments can be used for IO\$\_DEACCESS: the attribute list and the FIB\$L\_ACCTL field (used in that order); the FIB\$L\_ACCTL flag bits are ignored. The FIB\$W\_FID field can be nonzero. If so, it must match the file identifier of the accessed file. All other arguments are illegal and must be 0. IO\$\_DEACCESS takes no function modifiers.

## DISK, MAGNETIC TAPE, AND ACP QIO FUNCTIONS

### A.4 MODIFY FILE

This virtual I/O function modifies the file attributes and/or allocation of a disk file. If used with a magnetic tape file, MODIFY FILE is basically a NOP.

The function code is:

- IO\$MODIFY

The device/function-dependent arguments for IO\$MODIFY are:

- P1 -- the address of the File Information Block (FIB) descriptor.
- P2 -- the address of the file name string descriptor (optional). If specified, the name is entered in the directory specified by the FIB.
- P3 -- the address of the word that is to receive the length of the resultant file name string (optional).
- P4 -- the address of a descriptor for a buffer that is to receive the resultant file name string (optional).
- P5 -- the address of a list of attribute descriptors (optional). If specified, the indicated attributes are written to the file header.

An initial search is made for the indicated name string. The search is performed the same way, and with the same consequences, as the IO\$ACCESS search (see Section A.2). The file can be either extended or truncated. If extended (FIB\$M\_EXTEND is set), the amount is indicated by the extend control data (FIB\$L\_EXSZ) and the total number of blocks allocated to the file is returned in the second longword of the IO\$B. If truncated (FIB\$M\_TRUNC is set), the file is shortened to the number of blocks specified in FIB\$L\_EXVBN, minus 1. The resulting file size is returned in the second longword of the IO\$B.

The FIB\$W\_EXCTL field is interpreted as described in Table 9-1.

FIB\$L\_EXVBN and FIB\$L\_EXSZ are used to return the actual starting virtual block number (VBN) and size, respectively, of the area allocated or truncated.

The FIB\$W\_NMCTL and FIB\$L\_WCC fields are interpreted as described for IO\$ACCESS (see Section A.2). If an attribute list is present, the indicated file attributes are written. IO\$MODIFY takes no function modifiers.

Table A-3 lists the legal arguments for IO\$MODIFY in the order in which they are used. All other arguments are illegal and must be 0.

## DISK, MAGNETIC TAPE, AND ACP QIO FUNCTIONS

Table A-3  
IO\$\_MODIFY Arguments

FIB\$W_DID (disk only)
FIB\$W_NMCTL (disk only)
FIB\$L_WCC (disk only)
FIB\$M_EXTEND (disk only)
FIB\$L_EXSZ (disk only)
FIB\$W_EXCTL (disk only)
FIB\$M_TRUNC (disk only)
Attribute List

### A.5 DELETE FILE

This virtual I/O function removes a directory header and/or file header from a disk file.

The function code is:

- IO\$\_DELETE

The function modifier is:

- IO\$M\_DELETE

The device/function-dependent arguments for IO\$\_DELETE are:

- P1 -- the address of the File Information Block (FIB) descriptor.
- P2 -- the address of the file name string descriptor (optional). If specified, the name is entered in the directory specified by the FIB.
- P3 -- the address of the word that is to receive the length of the resultant file name string (optional).
- P4 -- the address of a descriptor for a buffer that is to receive the resultant file name string (optional).
- P5 -- the address of a list of attribute descriptors (optional). If specified, the indicated attributes are written to the file header.

A search is made for the directory entry to be deleted. The search is performed the same way as the IO\$\_ACCESS search (see Section A.2). The directory entry is then removed. The function modifier (IO\$M\_DELETE) deletes the file header specified by FIB\$W\_FID.

The only arguments for IO\$\_DELETE are FIB\$W\_DID and IO\$M\_DELETE, used in that order. All other arguments are illegal and must be 0.

## DISK, MAGNETIC TAPE, AND ACP QIO FUNCTIONS

### A.6 MOUNT

This virtual I/O function mounts a disk or magnetic tape volume. Mount privilege is required, IO\$\_MOUNT takes no arguments or function modifiers.

### A.7 ACP CONTROL

This virtual I/O function performs miscellaneous control functions, depending on the specified argument. If the function modifier IO\$\_DMOUNT is not set, the FIB control function field (FIB\$\_CNTRLFUNC) has one of its field values set. Listed below are the legal arguments for IO\$\_ACPCONTROL in the order in which they are used; all other arguments are illegal and must be 0:

IO\$\_DMOUNT

FIB\$\_CNTRLFUNC<sup>1</sup> field values:

FIB\$\_REWINDFIL

FIB\$\_POSEND

FIB\$\_NEXTVOL

FIB\$\_SPACE

FIB\$\_REWINDVOL

FIB\$\_CNTRLVAL<sup>1</sup>

<sup>1</sup>Only for magnetic tape devices



APPENDIX B  
I/O FUNCTION CODES

This appendix lists the function codes and function modifiers defined in the \$IODEF macro. The arguments for these functions are also listed.

B.1 TERMINAL DRIVER

<u>Function</u>	<u>Arguments</u>	<u>Modifier</u>
IO\$ READVBLK	P1 - buffer address	IO\$M_NOECHO
IO\$ READLBLK	P2 - buffer size	IO\$M_CVTLOW
IO\$ READPBLK	P3 - timeout	IO\$M_NOFILTR
IO\$ READPROMPT	P4 - read terminator block address	IO\$M_TIMED
	P5 - prompt string buffer address <sup>1</sup>	IO\$M_PURGE
	P6 - prompt string buffer size <sup>1</sup>	IO\$M_DSABLMBX
		IO\$M_TRMNOECHO
IO\$ WRITEVBLK	P1 - buffer address	IO\$M_CANCTRL0
IO\$ WRITELBLK	P2 - buffer size	IO\$M_ENABLMBX
IO\$ WRITEPBLK	P3 - (ignored)	IO\$M_NOFORMAT
	P4 - carriage control specifier <sup>2</sup>	
IO\$ SETMODE	P1 - characteristics buffer address	
IO\$ SETCHAR	P2 - (ignored)	
	P3 - speed specifier	
	P4 - fill specifier	
	P5 - parity flags	
IO\$ SETMODE!IO\$M_HANGUP	(none)	
IO\$ SETCHAR!IO\$M_HANGUP		
IO\$ SETMODE!IO\$M_CTRLCAST	P1 - AST service routine address	
IO\$ SETMODE!IO\$M_CTRLYAST	P2 - AST parameter	
IO\$ SETCHAR!IO\$M_CTRLCAST	P3 - access mode to deliver AST	
IO\$ SETCHAR!IO\$M_CTRLYAST		

<sup>1</sup>Only for IO\$ READPROMPT

<sup>2</sup>Only for IO\$ WRITELBLK and IO\$ WRITEVBLK

## I/O FUNCTION CODES

### B.2 DISK DRIVERS

<u>Functions</u>	<u>Arguments</u>	<u>Modifiers</u>
IO\$ READVBLK	P1 - buffer address	IO\$M_DATACHECK
IO\$ READLBLK	P2 - byte count	IO\$M_INHRETRY
IO\$ READPBLK	P3 - disk address	IO\$M_INHSEEK1
IO\$ WRITEVBLK		
IO\$ WRITELBLK		
IO\$ WRITEPBLK		
IO\$ SETMODE	P1 - characteristic buffer address	IO\$ INHRETRY
IO\$ SETCHAR		
IO\$ CREATE	P1 - FIB descriptor address	IO\$M_CREATE <sup>2</sup>
IO\$ ACCESS	P2 - file name string address	IO\$M_ACCESS <sup>2</sup>
IO\$ DEACCESS		IO\$M_DELETE <sup>3</sup>
IO\$ MODIFY	P3 - result string length address	
IO\$ DELETE	P4 - result string descriptor address	
	P5 - attribute list address	

<sup>1</sup>Only for IO\$ READPBLK and IO\$ WRITEPBLK

<sup>2</sup>Only for IO\$ CREATE and IO\$ ACCESS

<sup>3</sup>Only for IO\$ CREATE and IO\$ DELETE

### B.3 MAGNETIC TAPE DRIVERS

<u>Functions</u>	<u>Arguments</u>	<u>Modifiers</u>
IO\$ READVBLK	P1 - buffer address	IO\$M_DATACHECK
IO\$ READLBLK	P2 - byte count	IO\$M_INHRETRY
IO\$ READPBLK		IO\$M_REVERSE <sup>1</sup>
IO\$ WRITEVBLK		IO\$M_INHEXTGAP <sup>2</sup>
IO\$ WRITELBLK		
IO\$ WRITEPBLK		
IO\$ SETMODE	P1 - characteristics buffer address	IO\$M_INHRETRY
IO\$ SETCHAR		IO\$M_INHEXTGAP
IO\$ CREATE	P1 - FIB descriptor address	IO\$M_CREATE <sup>3</sup>
IO\$ ACCESS	P2 - file name string address	IO\$M_ACCESS <sup>3</sup>
IO\$ DEACCESS		IO\$M_DMOUNT <sup>4</sup>
IO\$ MODIFY	P3 - result string length address	
IO\$ ACPCONTROL	P4 - result string descriptor address	
	P5 - attribute list address	
IO\$ SKIPFILE	P1 - skip n tape marks	IO\$M_INHRETRY
IO\$ SKIPRECORD	P1 - skip n records	IO\$M_INHRETRY

<sup>1</sup>Only for read functions

<sup>2</sup>Only for write Functions

<sup>3</sup>Only for IO\$ CREATE and IO\$ ACCESS

<sup>4</sup>Only for IO\$ ACPCONTROL

## I/O FUNCTION CODES

<u>Functions</u>	<u>Arguments</u>	<u>Modifiers</u>
IO\$_MOUNT	(none)	
IO\$_REWIND IO\$_REWINDOFF	(none)	IO\$_INHRETRY IO\$_NOWAIT
IO\$_WRITEOF	(none)	IO\$_INHEXTGAP IO\$_INHRETRY
IO\$_SENSEMODE	(none)	IO\$_INHRETRY

<sup>1</sup>Only for read functions

<sup>2</sup>Only for write functions

<sup>3</sup>Only for IO\$\_CREATE and IO\$\_ACCESS

<sup>4</sup>Only for IO\$\_ACPCONTROL

### B.4 LINE PRINTER DRIVER

<u>Functions</u>	<u>Arguments</u>	<u>Modifiers</u>
IO\$_WRITEVBLK IO\$_WRITELBLK IO\$_WRITEPBLK	P1 - buffer address P2 - buffer size P3 - (ignored) P4 - carriage control specifier <sup>1</sup>	(none)
IO\$_SETMODE IO\$_SETCHAR	P1 - characteristics buffer address	(none)

<sup>1</sup>Only for IO\$\_WRITEVBLK and IO\$\_WRITELBLK

### B.5 CARD READER DRIVER

<u>Functions</u>	<u>Arguments</u>	<u>Modifiers</u>
IO\$_READLBLK IO\$_READVBLK IO\$_READPBLK	P1 - buffer address P2 - byte count	IO\$_BINARY IO\$_PACKED
IO\$_SETMODE IO\$_SETCHAR	P1 - characteristics buffer address	(none)
IO\$_SENSEMODE	(none)	

### B.6 MAILBOX DRIVER

<u>Functions</u>	<u>Arguments</u>	<u>Modifiers</u>
IO\$_READVBLK IO\$_READLBLK IO\$_READPBLK IO\$_WRITEVBLK IO\$_WRITELBLK IO\$_WRITEPBLK	P1 - buffer address P2 - buffer size	IO\$_NOW
IO\$_WRITEOF	(none)	
IO\$_SETMODE!IO\$_READATTN IO\$_SETMODE!IO\$_WRTATTN	P1 - AST address P1 - AST parameter	

## I/O FUNCTION CODES

### B.7 DMC11 DRIVER

<u>Functions</u>	<u>Arguments</u>	<u>Modifiers</u>
IO\$ READLBLK	P1 - buffer address	IO\$M_DSABLMBX <sup>2</sup>
IO\$ READPBLK	P2 - message size	IO\$M_NOW <sup>2</sup>
IO\$ READVBLK	P6 - diagnostic buffer <sup>1</sup>	IO\$M_ENABLMBX <sup>3</sup>
IO\$ WRITELBLK		
IO\$ WRITEPBLK		
IO\$ WRITEVBLK		
IO\$ SETMODE	P1 - characteristics	
IO\$ SETCHAR	buffer address	
IO\$ SETMODE!IO\$M_ATTNAST	P1 - AST service	
IO\$ SETCHAR!IO\$M_ATTNAST	routine address	
	P2 - (ignored)	
	P3 - AST access mode	
IO\$ SETMODE!IO\$M_SHUTDOWN	P1 - characteristics	
IO\$ SETCHAR!IO\$M_SHUTDOWN	block address	
IO\$ SETMODE!IO\$M_STARTUP	P1 - characteristics	
IO\$ SETCHAR!IO\$M_STARTUP	block address	
	P2 - (ignored)	
	P3 - receive message	
	blocks	

<sup>1</sup>Only for IO\$ READPBLK and IO\$ WRITEPBLK

<sup>2</sup>Only for IO\$ READLBLK and IO\$ READPBLK

<sup>3</sup>Only for IO\$ WRITELBLK and IO\$ WRITEPBLK

### B.8 ACP INTERFACE DRIVER

<u>Functions</u>	<u>Arguments</u>	<u>Modifiers</u>
IO\$ CREATE	P1 - FIB descriptor address	IO\$M_CREATE <sup>1</sup>
IO\$ ACCESS	P2 - file name string	IO\$M_ACCESS <sup>1</sup>
IO\$ DEACCESS	address	IO\$M_DELETE <sup>2</sup>
IO\$ MODIFY	P3 - result string length	IO\$M_DMOUNT <sup>3</sup>
IO\$ DELETE	address	
IO\$ ACPCONTROL	P4 - result string descriptor	
	address	
	P5 - attribute list address	
IO\$ MOUNT	(none)	

<sup>1</sup>Only for IO\$ CREATE and IO\$ ACCESS

<sup>2</sup>Only for IO\$ CREATE and IO\$ DELETE

<sup>3</sup>Only for IO\$ ACPCONTROL

## INDEX

\$ASSIGN, 1-13, 2-3, 7-2, 8-2  
\$CREMBX, 1-14, 7-2  
\$GETCHN, 1-24  
\$GETDEV, 1-24  
\$INPUT, 1-19  
\$OUTPUT, 1-19  
\$QIO and \$QIOW device/function  
    independent arguments, 1-16  
\$QIO macro, 1-15  
\$QIOW macro, 1-16  
\$WAITFR, 1-15

026 code, 6-2  
029 code, 6-2

8-bit ASCII, 2-13

### A

Access, 1-5, 1-6, 1-14  
Access file, A-2  
ACP functions, 9-1  
ACP interface driver I/O  
    functions, B-4  
ACP QIO functions, 9-1, 9-9  
    arguments, 9-2  
    attributes, 9-12  
    disk magnetic tape and, A-1  
    function modifiers, 9-2  
    status returns, 9-14  
Allocate contiguous space, 9-7  
Allocate Device (\$ALLOC) system  
    service, 1-14  
ALTMODE, 2-8  
Ancillary control process (ACP),  
    9-1  
Arguments, 1-15  
    ACP QIO functions, 9-2  
    device/function dependent,  
        1-18  
    device/function independent,  
        1-16  
    I/O function codes, B-1  
Assign I/O Channel (\$ASSIGN)  
    system service, 1-13, 7-2  
Assigning channels, 1-13  
AST address, 1-18  
AST parameter, 1-18  
AST quota, 1-4, 7-5  
Asynchronous System Traps, 1-23

Attention AST,  
    enable DMCl1, 8-8  
    read, 7-7  
    write, 7-7  
Attribute control block, 9-11  
Attributes, ACP QIO, 9-12

### B

Beginning-of-tape (BOT), 4-10  
Block-addressable devices, 1-8,  
    1-10  
Buffered I/O byte count quota,  
    1-4  
Buffered I/O quota, 1-3, 7-5

### C

CALL, 1-23  
Card punch combinations, 6-2  
Card reader,  
    device characteristics, 6-4  
    end-of-file, 6-2  
    I/O functions, 6-5, B-3  
    I/O status block, 6-8  
    read function, 6-6  
    set characteristic, 6-7  
    set mode, 6-7  
    status returns, 6-8  
    translation mode, 6-2  
Carriage control,  
    line printer, 5-5  
    terminal, 2-17  
Channels, 1-13  
Channel assignments, 1-13  
Channel number, 1-17  
Character bit mask,  
    terminator, 2-16  
Character formatting, line  
    printer, 5-2  
Character interpretation, 2-8  
Characteristics (see Device  
    characteristics)  
Close check, 9-5  
Completion status, 1-19, 1-22  
Console terminal, 2-1  
Contiguous space,  
    allocate, 9-7  
Control characters,  
    terminal, 2-5  
Create file, A-1  
Create Mailbox and Assign  
    Channel (\$CREMBX) system  
    service, 1-14, 7-2

## INDEX (Cont.)

CTRL/C, 2-5, 2-23  
CTRL/C AST,  
    enable, 2-23  
CTRL/I, 2-5  
CTRL/J, 2-5  
CTRL/K, 2-6  
CTRL/L, 2-6  
CTRL/O, 2-6  
CTRL/Q, 2-6, 2-12  
CTRL/R, 2-6  
CTRL/S, 2-6, 2-12  
CTRL/U, 2-7  
CTRL/X, 2-7  
CTRL/Y, 2-7, 2-23  
CTRL/Y AST,  
    enable, 2-23  
CTRL/Z, 2-7

## D

Data check,  
    disk, 3-2, 3-9, 3-10  
    magnetic tape, 4-2, 4-9, 4-10  
Data interpretation, 2-8, 2-13  
DDCMP, 8-1, 8-4  
Deaccess file, A-4  
Deaccess lock, 9-5  
Deassign I/O channel (\$DEASSGN)  
    system service, 7-3  
DELETE, 2-8  
Delete file, A-6  
Delete mailbox (\$DELMBX) system  
    service, 7-3  
Device allocation, 1-14  
Device characteristics,  
    card reader, 6-3  
    disk, 3-4  
    DMC11, 8-3  
    line printer, 5-3, 5-4  
    magnetic tape, 4-3  
    mailbox, 7-4  
    terminal, 2-10  
Device information, 1-23  
Device-dependent  
    characteristics, line  
    printer, 5-4  
Device-independent  
    characteristics, line  
    printer, 5-3  
Device/function dependent  
    arguments, 1-18  
Device/function dependent bits,  
    1-13  
Device/function independent  
    bits, 1-13  
Devices, 1-1  
Dial-in message, 2-10  
Dial-up, 2-10, 2-13, 2-23

DIGITAL data communications  
    message protocol (DDCMP),  
    8-1  
Direct I/O quota, 1-4, 8-3  
Directory file, 9-6  
Disk, magnetic tape, and ACP  
    QIO functions, A-1  
Disk,  
    device characteristics, 3-4  
    devices, 3-1  
    drivers, 3-1  
    error recovery, 3-4, 3-9,  
    3-10  
    I/O functions, 3-6, B-2  
    I/O status block, 3-12  
    read function, 3-9  
    set characteristic, 3-11  
    set mode, 3-10  
    status returns, 3-12  
    write function, 3-10  
DMC11 synchronous  
    communications line  
    interface driver, 8-1  
DMC11,  
    device characteristics, 8-3  
    enable attention AST, 8-8  
    error summary bits, 8-6  
    I/O functions, 8-6, B-4  
    I/O status block, 8-10  
    mailbox usage, 8-2  
    message size, 8-4  
    read function, 8-6  
    set characteristics, 8-8  
    set mode, 8-7, 8-8  
    shut down unit, 8-9  
    start unit, 8-9  
    status returns, 8-10  
    unit characteristics, 8-5  
    write function, 8-7  
DZ-11 Asynchronous Serial Line  
    Multiplexer, 2-1

## E

ECC correction, 3-3  
Enable attention AST,  
    DMC11, 8-9  
    mailbox, 7-7  
Enable CTRL/C AST, 2-23  
Enable CTRL/Y AST, 2-23  
End-of-file, card reader, 6-2  
End-of-file message, write, 7-7  
End-of-file status, 4-9  
End-of-tape status, 4-9, 4-10  
Error recovery,  
    disk devices, 3-3, 3-9, 3-10  
    line printer, 5-2  
    magnetic tape, 4-3

## INDEX (Cont.)

Error severity level, 1-20  
Error summary bits, DMC11, 8-6  
ESCAPE, 2-8  
Escape sequences, 2-2, 2-12  
Event flags, 1-15, 1-22  
Event flag number, 1-17  
Extend control, 9-7

## F

FIB argument usage, 9-9  
FIB fields, 9-3  
File attributes, 9-11  
File identification, 9-6  
File Information Block (FIB),  
9-3  
Fill specifier, 2-21  
Foreign volume, 1-10  
FORM FEED, 2-6  
Form feeds, 2-17, 5-4  
Free list,  
receive buffer, 8-3, 8-10  
Full-duplex, 2-10  
Function codes, 1-12, 1-17, B-1  
Function modifiers, 1-12, B-1

## G

Get Channel Information  
(\$GETCHN) system service,  
1-23  
Get Device Information  
(\$GETDEV) system service,  
1-23

## H

Half-duplex, 2-10  
Hang-up, 2-10, 2-13  
Hang-up function modifier, 2-23  
Hang-up,  
terminal, 2-4  
Holdscreen Mode, 2-12  
Host/terminal synchronization,  
2-12

## I

I/O completion, 1-21  
I/O function code,  
arguments, B-1  
I/O function codes, 1-12, B-1  
I/O functions, 1-6, 1-12, B-1  
card reader, 6-5  
disk, 3-6

I/O functions (Cont.),  
DMC11, 8-6  
line printer, 5-4  
magnetic tape, 4-5  
mailbox, 7-5  
terminal, 2-13  
I/O operations, 1-6  
logical, 1-8  
physical, 1-6  
virtual, 1-10  
I/O quota,  
buffered, 1-3, 7-5  
byte count, 1-4  
direct, 1-4, 8-3  
I/O requests, 1-1, 1-13, 1-15  
I/O status block, 1-15, 1-18,  
1-22  
ACP QIO interface, 9-14  
card reader, 6-8  
disk devices, 3-12  
DMC11, 8-10  
line printer, 5-9  
magnetic tape devices, 4-14  
mailbox, 7-9  
terminal, 2-24  
I/O status returns, 1-21  
I/O system services, 1-2  
Inhibit retry, 3-9, 3-10, 4-6,  
4-7  
Input/output operations, 1-1  
Interactive terminal, 2-5  
IO\$M\_ACCESS, 3-7, 4-6, 9-2  
IO\$M\_BINARY, 6-1  
IO\$M\_CREATE, 3-7, 4-6, 9-2  
IO\$M\_DATACHECK, 3-7, 4-6  
IO\$M\_DELETE, 3-7, 9-2  
IO\$M\_DMOUNT, 4-7, 9-2  
IO\$M\_INHERLOG, 1-8  
IO\$M\_INHEXTGAP, 4-6  
IO\$M\_INHRETRY, 3-7, 4-6  
IO\$M\_INHSEEK, 3-7  
IO\$M\_NOWAIT, 4-7  
IO\$M\_PACKED, 6-1  
IO\$M\_REVERSE, 4-6  
IO\$ \_ACCESS, 3-7, 4-6, 9-9, 9-10  
IO\$ \_ACPCONTROL, 4-7, 9-11  
IO\$ \_CREATE, 3-7, 4-6, 9-9  
IO\$ \_DEACCESS, 3-7, 4-6  
IO\$ \_DELETE, 3-7, 9-10  
IO\$ \_MODIFY, 3-7, 4-6, 9-10  
IO\$ \_MOUNT, 4-7, 9-2  
IO\$B, 1-22  
Issuing QIO requests, 1-13

## K

Keywords, 1-15

## INDEX (Cont.)

### L

LINE FEED, 2-5  
Line feeds, 2-17  
Line printer,  
  carriage control, 5-5  
  character formatting, 5-1  
  device-dependent  
    characteristics, 5-4  
  device-independent  
    characteristics, 5-3  
  driver, 5-1  
  error recovery, 5-2  
  I/O functions, 5-4, B-3  
  I/O status block, 5-9  
  set characteristics, 5-8  
  set mode, 5-8  
  status returns, 5-9  
  write function, 5-4  
Line,  
  remote, 2-4, 2-10, 2-13  
Logical I/O operations, 1-8  
Logical I/O privilege, 1-4, 1-6,  
  1-8  
Logical name, 1-14  
Lowercase, 2-12, 2-17, 5-2, 5-4

### M

Magnetic tape and ACP QIO  
  functions, A-1  
Magnetic tape,  
  device characteristics, 4-3  
  driver, 4-1  
  error recovery, 4-3  
  I/O functions, 4-5, B-2  
  I/O status block, 4-14  
  read function, 4-8  
  set characteristics, 4-12  
  set mode, 4-12  
  status returns, 4-14  
  write function, 4-10  
Mailbox,  
  creation, 1-14  
  device characteristics, 1-24,  
    7-4  
  driver, 7-1  
  I/O functions, 7-5, B-3  
  I/O status block, 7-9  
  message format, 7-3  
  protection, 1-5  
  QIO requests, 7-5, 7-6  
  read attention AST, 7-7  
  set attention AST QIO  
    requests, 7-7  
  status returns, 7-9  
  terminal, 2-3  
  usage (DMC11), 8-2

Mailbox (Cont.),  
  write attention AST, 7-7  
  write end-of-file message, 7-7  
Master adapter, 4-2  
Mechanical form feed, 5-4  
Mechanical tabs, 2-12  
Message format, mailbox, 7-3  
Message size, DMC11, 8-4  
Modem control, 2-1, 2-10  
Modify file, A-5  
MOUNT, 1-14  
Mount privilege, 1-5  
Mounted foreign, 1-8, 1-10, 3-9,  
  3-10

### N

Name string, 9-6  
Network, 1-24

### O

Offset recovery, 3-3

### P

Page length, 2-11  
Page width, 2-11  
Parity flags, terminal, 2-22  
PASSALL, 2-12, 2-13, 2-15  
Physical device name, 1-14  
Physical I/O operation, 1-6  
Physical I/O privilege, 1-4,  
  1-6, 1-8  
Printer (see Line printer)  
Privilege, 1-3  
  logical I/O, 1-4, 1-6, 1-8  
  mount, 1-5  
  physical I/O, 1-4, 1-6, 1-8  
Prompt buffer, terminal, 2-14  
Protection, 1-3, 1-5  
Protection mask, 1-5, 1-6, 1-8,  
  1-10  
Protocol (DDCMP), 8-1, 8-9

### Q

QIO interface to ACPs, 9-1  
QIO macro, 1-15  
QIOW macro, 1-16  
Queue I/O (\$QIO) system service,  
  1-1, 1-14  
Queue I/O Request and Wait For  
  Event Flag (\$QIOW) system  
  service, 1-16  
Quotas, 1-3, 3-6, 4-5, 7-5, 8-3

INDEX (Cont.)

**R**

Read access, 9-5  
 Read attention AST, 7-7  
 Read binary, 6-1, 6-6  
 Read checking, 9-5  
 Read function,  
   card reader, 6-6  
   disk devices, 3-9  
   DMC11, 8-6  
   magnetic tape devices, 4-8  
   terminal, 2-14  
 Read mailbox QIO requests, 7-5  
 Read packed Hollerith, 6-1, 6-6  
 Read QIO function, 1-6  
 Read with prompt, 2-14  
 Read with timeout, 2-14  
 Receive buffer free list, 8-3,  
   8-10  
 Receive-message blocks, 8-10  
 Record-oriented devices, 1-8,  
   1-10  
 Remote line, 2-4, 2-10, 2-13  
 Resource wait mode, 1-3, 7-2  
 RETURN, 2-8  
 Rewind offline, 4-12  
 RMS, 1-1

**S**

Seek operations, 3-3  
 Sense card reader mode, 6-7  
 Sense tape mode, 4-12  
 Set attention AST QIO requests,  
   7-7, 8-8  
 Set characteristics,  
   card reader, 6-7  
   disk devices, 3-10  
   DMC11, 8-7  
   line printer, 5-8  
   magnetic tape devices, 4-12  
   terminal, 2-21  
 Set mode QIO function, 1-6  
 Set mode,  
   card reader, 6-7  
   disk devices, 3-10  
   DMC11, 8-7  
   line printer, 5-8  
   magnetic tape devices, 4-12  
   terminal, 2-21  
 Set Resource Wait Mode (\$SETRWM)  
   system service, 1-3  
 Set Terminal command, 2-11  
 Severity level,  
   error, 1-20  
 Shut down unit (DMC11), 8-9  
 Skip file, 4-11  
 Skip record, 4-11

Slave formatter, 4-2  
 Software channels, 1-1  
 Software mailboxes, 7-1  
 Spaces (terminal), 2-17  
 Speed specifier (terminal),  
   2-21  
 Spooled device characteristics,  
   1-24  
 SS\$ ABORT, 2-25, 5-9, 8-11  
 SS\$ ACCONFLICT, 9-15  
 SS\$ ACCVIO, 1-20, 1-26  
 SS\$ ACPVAFUL, 9-15  
 SS\$ BADATTRIB, 9-15  
 SS\$ BADCHKSUM, 9-15  
 SS\$ BADESCAPE, 2-25  
 SS\$ BADFILEHDR, 9-15  
 SS\$ BADFILENAME, 9-15  
 SS\$ BADFILEVER, 9-15  
 SS\$ BADIRECTORY, 9-15  
 SS\$ BADPARAM, 9-15  
 SS\$ BLOCKCNTERR, 9-15  
 SS\$ BUFFEROVF, 1-26  
 SS\$ CONTROLC, 2-26  
 SS\$ CONTROLLO, 2-25  
 SS\$ CONTROLY, 2-26  
 SS\$ CREATED, 9-15  
 SS\$ CTRLERR, 3-12, 4-14  
 SS\$ DATACHECK, 3-12, 4-15  
 SS\$ DATAOVERUN, 4-16, 6-9, 8-11  
 SS\$ DEVACTIVE, 8-11  
 SS\$ DEVFOREIGN, 1-10  
 SS\$ DEVICEFULL, 9-15  
 SS\$ DEVNOTMOUNT, 1-10  
 SS\$ DEVOFFLINE, 8-11  
 SS\$ DIRFULL, 9-15  
 SS\$ DRVERR, 3-12, 4-15  
 SS\$ DUPFILENAME, 9-16  
 SS\$ ENDOFFILE, 4-15, 6-9, 7-10,  
   8-11, 9-16  
 SS\$ ENDOFTAPE, 4-15  
 SS\$ EXQUOTA, 1-20  
 SS\$ FCPREADERR, 9-16  
 SS\$ FCPREWINDERR, 9-16  
 SS\$ FCPSPACERR, 9-16  
 SS\$ FCPWRITERR, 9-16  
 SS\$ FILELOCKED, 9-16  
 SS\$ FILENUMCHK, 9-16  
 SS\$ FILESEQCHK, 9-16  
 SS\$ FILESTRUCT, 9-16  
 SS\$ FILNOTEXP, 9-16  
 SS\$ FORMAT, 3-13, 4-15  
 SS\$ HEADERFULL, 9-16  
 SS\$ IDXFILEFULL, 9-16  
 SS\$ ILLCNTRFUNC, 9-16  
 SS\$ ILLEFC, 1-20  
 SS\$ INSMEM, 1-20  
 SS\$ IVADDR, 3-13  
 SS\$ IVCHAN, 1-21, 1-26  
 SS\$ MEDOFL, 3-13, 4-15

## INDEX (Cont.)

SS\$ \_NOMOREFILES, 9-17  
 SS\$ \_NONEXDRV, 3-13, 4-15  
 SS\$ \_NOPRIV, 1-6, 1-8, 1-10,  
     1-21, 1-26, 9-17  
 SS\$ \_NORMAL, 1-20, 1-26, 2-25,  
     3-12, 4-14, 5-9, 6-9, 7-10,  
     8-11  
 SS\$ \_NOSUCHFILE, 9-17  
 SS\$ \_NOTAPEOP, 9-17  
 SS\$ \_NOTLABELMT, 9-17  
 SS\$ \_PARITY, 2-26, 3-13, 4-16  
 SS\$ \_PARTEscape, 2-25  
 SS\$ \_SUPERSEDE, 9-17  
 SS\$ \_TAPEPOSLOST, 9-17  
 SS\$ \_TIMEOUT, 2-25  
 SS\$ \_TOOMANYVER, 9-17  
 SS\$ \_UNASEFC, 1-21  
 SS\$ \_UNSAFE, 3-13, 4-16  
 SS\$ \_VOLINV, 3-13, 4-16  
 SS\$ \_WASECC, 3-13  
 SS\$ \_WRITLCK, 3-13, 4-16  
 SS\$ \_WRTLCK, 9-17  
 Start unit (DMC11), 8-9  
 Status returns,  
     ACP QIO interface, 9-14  
     card reader, 6-8  
     disk devices, 3-12  
     DMC11, 8-10  
     I/O, 1-21  
     line printer, 5-9  
     magnetic tape devices, 4-14  
     mailbox, 7-9  
     system services, 1-20  
     terminal, 2-24  
 Status,  
     completion, 1-19, 1-22  
 System services, I/O, 1-2  
 System services status returns,  
     1-20

## T

TAB, 2-5  
 Tabs, 2-12, 2-17  
 Tape (see Magnetic tape)  
 Terminal,  
     carriage control, 2-17  
     characteristics, 2-10  
     control characters, 2-5  
     driver, 2-1  
     enable CTRL/C AST, 2-23  
     enable CTRL/Y AST, 2-23  
     function modifiers, 2-15  
     hang-up, 2-4  
     hang-up function modifier,  
         2-23  
     I/O functions, 2-13, B-1  
     I/O status block, 2-24

Terminal (Cont.),  
     mailbox, 2-3  
     read function, 2-14  
     read terminator set, 2-16  
     set characteristic, 2-20  
     set mode, 2-20  
     status returns, 2-24  
     write function, 2-16  
 Terminal/host synchronization,  
     2-12  
 Terminator character bit mask,  
     2-16  
 Terminator set, 2-16  
 Transfer count, I/O, 1-22  
 Translation mode, card reader,  
     6-2  
 Truncation, 9-7, 9-14  
 Type-ahead, 2-1, 2-4, 2-13, 2-15

## U

Unsolicited data, 2-4  
 Uppercase, 2-12, 2-17, 5-2, 5-4

## V

VAX-11 Record Management  
     Services (RMS), 1-1  
 VAX/VMS System Services  
     Reference Manual, 1-3  
 Version number, 9-6  
 VERTICAL TAB, 2-6  
 Virtual I/O operations, 1-10  
 Volume protection, 1-5

## W

Wait for Single Event Flag  
     (\$WAITFR) system service,  
         1-15  
 Wild card directory, 9-6  
 Write access, 9-5  
 Write attention AST, 7-7  
 Write checking, 9-5  
 Write end-of-file, 4-11  
 Write end-of-file message, 7-7  
 Write function,  
     disk, 3-10  
     DMC11, 8-7  
     line printer, 5-4  
     magnetic tape, 4-10  
     terminal, 2-16  
 Write mailbox QIO requests, 7-6  
 Write QIO function, 1-6



Do Not Tear - Fold Here and Tape

**digital**



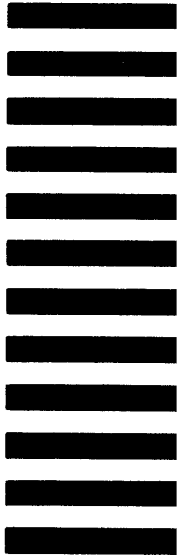
No Postage  
Necessary  
if Mailed in the  
United States

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

RT/C SOFTWARE PUBLICATIONS TW/A14  
DIGITAL EQUIPMENT CORPORATION  
1925 ANDOVER STREET  
TEWKSBURY, MASSACHUSETTS 01876



Do Not Tear - Fold Here