

VMS I/O User's Reference Manual: Part I

Order Number: AA-LA84A-TE

April 1988

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

Revision/Update Information: This document supersedes the *VAX/VMS I/O User's Reference Manual: Part I, Version 4.4.*

Software Version: VMS Version 5.0

**digital equipment corporation
maynard, massachusetts**

April 1988

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

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


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

Copyright ©1988 by Digital Equipment Corporation

All Rights Reserved.
Printed in U.S.A.

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

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem-10	PDP	VT
DECSYSTEM-20	PDT	
DECUS	RSTS	
DECwriter	RSX	

ZK4513

**HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS**

USA & PUERTO RICO*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire
03061

CANADA

Digital Equipment
of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

INTERNATIONAL

Digital Equipment Corporation
PSG Business Manager
c/o Digital's local subsidiary
or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

* Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminister, Massachusetts 01473.

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript[™] printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

Contents

PREFACE	xix
NEW AND CHANGED FEATURES	xxiii
CHAPTER 1 ACP—QIO INTERFACE	1-1
1.1 ACP FUNCTIONS AND ENCODING	1-2
1.2 FILE INFORMATION BLOCK (FIB)	1-3
1.3 ACP SUBFUNCTIONS	1-8
1.3.1 Directory Lookup	1-9
1.3.1.1 Input Parameters • 1-9	
1.3.1.2 Operation • 1-10	
1.3.1.3 Directory Entry Protection • 1-11	
1.3.2 Access	1-11
1.3.2.1 Input Parameters • 1-11	
1.3.2.2 Operation • 1-13	
1.3.3 Extend	1-13
1.3.3.1 Input Parameters • 1-13	
1.3.3.2 Operation • 1-15	
1.3.4 Truncate	1-15
1.3.4.1 Input Parameters • 1-16	
1.3.4.2 Operation • 1-16	
1.3.5 Read/Write Attributes	1-17
1.3.5.1 Input Parameters • 1-17	
1.4 ACP QIO RECORD ATTRIBUTES AREA	1-21
1.5 ACP-QIO ATTRIBUTES STATISTICS BLOCK	1-23
1.6 MAJOR FUNCTIONS	1-25
1.6.1 Create File	1-25
1.6.1.1 Input Parameters • 1-25	
1.6.1.2 Disk ACP Operation • 1-27	
1.6.1.3 Directory Entry Creation • 1-28	
1.6.1.4 Magnetic Tape ACP Operation • 1-28	
1.6.2 Access File	1-29

Contents

1.6.2.1	Input Parameters • 1–29	
1.6.2.2	Operation • 1–31	
1.6.3	Deaccess File _____	1–31
1.6.3.1	Input Parameters • 1–31	
1.6.3.2	Operation • 1–32	
1.6.4	Modify File _____	1–32
1.6.4.1	Input Parameters • 1–32	
1.6.4.2	Operation • 1–33	
1.6.5	Delete File _____	1–33
1.6.5.1	Operation • 1–34	
1.6.6	Mount _____	1–34
1.6.7	ACP Control _____	1–34
1.6.7.1	Input Parameters • 1–35	
1.6.7.2	Magnetic Tape Control Functions • 1–36	
1.6.7.3	Miscellaneous Disk Control Functions • 1–36	
1.6.7.4	Disk Quotas • 1–37	
<hr/>		
1.7	I/O STATUS BLOCK	1–40
<hr/>		
CHAPTER 2	CARD READER DRIVER	2–1
<hr/>		
2.1	SUPPORTED CARD READER DEVICE	2–1
<hr/>		
2.2	DRIVER FEATURES	2–1
2.2.1	Special Card Punch Combinations _____	2–1
2.2.1.1	End-of-File Condition • 2–2	
2.2.1.2	Set Translation Mode • 2–2	
2.2.2	Submitting Batch Jobs Through the Card Reader _____	2–2
2.2.3	Passing Data to Commands and Images _____	2–3
2.2.4	Error Recovery _____	2–4
<hr/>		
2.3	DEVICE INFORMATION	2–5
<hr/>		
2.4	CARD READER FUNCTION CODES	2–5
2.4.1	Read _____	2–6
2.4.2	Sense Mode _____	2–7
2.4.3	Set Mode _____	2–7
2.4.3.1	Set Mode • 2–8	
2.4.3.2	Set Characteristic • 2–10	
<hr/>		
2.5	I/O STATUS BLOCK	2–11

CHAPTER 3	DISK DRIVERS	3-1
<hr/>		
3.1	SUPPORTED DISK DEVICES AND CONTROLLERS	3-1
3.1.1	UDA50 UNIBUS Disk Adapter _____	3-2
3.1.2	KDA50 Disk Controller _____	3-3
3.1.3	KDB50 Disk Controller _____	3-3
3.1.4	HSC50 Controller _____	3-3
3.1.5	RA60 Pack Disk _____	3-4
3.1.6	RB02 and RL02 Cartridge Disks _____	3-4
3.1.7	RM03 and RM05 Pack Disks _____	3-4
3.1.8	RA80/R80/RM80 and RA81 Fixed Media Disks _____	3-4
3.1.9	RP05 and RP06 Pack Disks _____	3-5
3.1.10	RP07 Fixed Media Disk _____	3-5
3.1.11	RK06 and RK07 Cartridge Disks _____	3-5
3.1.12	RC25 Disk _____	3-5
3.1.13	RRD50 Read-Only Memory (CDROM) _____	3-5
3.1.14	RX01 Console Disk _____	3-6
3.1.15	RX02 Disk _____	3-6
3.1.16	TU58 Magnetic Tape (DECtape II) _____	3-7
<hr/>		
3.2	DRIVER FEATURES	3-7
3.2.1	Dual Porting (MASSBUS) _____	3-8
3.2.1.1	Port Selection and Access Modes • 3-8	
3.2.1.2	Disk Use and Restrictions • 3-9	
	3.2.1.2.1 Restriction on Dual-Ported Non-DSA Disks in a VAXcluster • 3-9	
3.2.1.3	Dual-Porting DSA Disks • 3-10	
3.2.2	Data Check _____	3-10
3.2.3	Overlapped Seeks _____	3-11
3.2.4	Error Recovery _____	3-11
3.2.4.1	Skip Sectoring • 3-12	
3.2.5	Logical-to-Physical Translation (RX01 and RX02) _____	3-12
3.2.6	DIGITAL Storage Architecture (DSA) Devices _____	3-13
3.2.6.1	Bad Block Replacement and Forced Errors for DSA Disks • 3-14	
3.2.7	VAXstation 2000 and MicroVAX 2000 Disk Driver _____	3-15
<hr/>		
3.3	DEVICE INFORMATION	3-16
<hr/>		
3.4	DISK FUNCTION CODES	3-17
3.4.1	Read _____	3-22
3.4.2	Write _____	3-23
3.4.3	Sense Mode _____	3-24
3.4.4	Set Density _____	3-24

Contents

3.4.5	Search _____	3-25
3.4.6	Pack Acknowledge _____	3-25
3.4.7	Unload _____	3-25
3.4.8	Available _____	3-26
3.4.9	Seek _____	3-26
3.4.10	Write Check _____	3-26
<hr/>		
3.5	I/O STATUS BLOCK	3-27
<hr/>		
3.6	PROGRAMMING EXAMPLE	3-27
<hr/>		
CHAPTER 4	LABORATORY PERIPHERAL ACCELERATOR DRIVER	4-1
<hr/>		
4.1	SUPPORTED DEVICE	4-1
4.1.1	LPA11-K Modes of Operation _____	4-1
4.1.2	Errors _____	4-2
<hr/>		
4.2	SUPPORTING SOFTWARE	4-3
<hr/>		
4.3	DEVICE INFORMATION	4-4
<hr/>		
4.4	LPA11-K FUNCTION CODES	4-7
4.4.1	Load Microcode _____	4-7
4.4.2	Start Microprocessor _____	4-8
4.4.3	Initialize LPA11-K _____	4-8
4.4.4	Set Clock _____	4-9
4.4.5	Start Data Transfer Request _____	4-10
4.4.6	LPA11-K Data Transfer Stop Command _____	4-13
<hr/>		
4.5	HIGH-LEVEL LANGUAGE INTERFACE	4-13
4.5.1	High-Level Language Support Routines _____	4-13
4.5.1.1	Buffer Queue Control • 4-14	
4.5.1.2	Subroutine Argument Usage • 4-15	
4.5.2	LPA\$ADSWP — Initiate Synchronous A/D Sampling Sweep _____	4-18
4.5.3	LPA\$DASWP — Initiate Synchronous D/A Sweep _____	4-19
4.5.4	LPA\$DISWP — Initiate Synchronous Digital Input Sweep _____	4-20
4.5.5	LPA\$DOSWP — Initiate Synchronous Digital Output Sweep _____	4-21
4.5.6	LPA\$LAMSKS — Set LPA11-K Masks and NUM Buffer _____	4-21
4.5.7	LPA\$SETADC — Set Channel Information for Sweeps _____	4-22

4.5.8	LPA\$SETIBF — Set IBUF Array for Sweeps _____	4-23
4.5.9	LPA\$STPSWP — Stop In-Progress Sweep _____	4-23
4.5.10	LPA\$CLOCKA — Clock A Control _____	4-24
4.5.11	LPA\$CLOCKB — Clock B Control _____	4-25
4.5.12	LPA\$XRATE — Compute Clock Rate and Preset Value _____	4-26
4.5.13	LPA\$IBFSTS — Return Buffer Status _____	4-26
4.5.14	LPA\$IGTBUF — Return Buffer Number _____	4-27
4.5.15	LPA\$INXTBF — Set Next Buffer to Use _____	4-28
4.5.16	LPA\$IWTBUF — Return Next Buffer or Wait _____	4-29
4.5.17	LPA\$RLSBUF — Release Data Buffer _____	4-29
4.5.18	LPA\$RMVBUF — Remove Buffer from Device Queue _____	4-30
4.5.19	LPA\$CVADF — Convert A/D Input to Floating-Point _____	4-31
4.5.20	LPA\$FLT16 — Convert Unsigned 16-bit Integer to Floating-Point _____	4-31
4.5.21	LPA\$LOADMC — Load Microcode and Initialize LPA11-K _____	4-31
<hr/>		
4.6	I/O STATUS BLOCK _____	4-32
<hr/>		
4.7	LOADING LPA11-K MICROCODE _____	4-33
4.7.1	Microcode Loader Process _____	4-33
4.7.2	Operator Process _____	4-34
<hr/>		
4.8	RSX-11M/M-PLUS AND VMS DIFFERENCES _____	4-34
4.8.1	General _____	4-34
4.8.2	Alignment and Length _____	4-35
4.8.3	Status Returns _____	4-35
4.8.4	Sweep Routines _____	4-35
<hr/>		
4.9	PROGRAMMING EXAMPLES _____	4-35
4.9.1	LPA11-K High-Level Language Program (Program A) _____	4-36
4.9.2	LPA11-K High-Level Language Program (Program B) _____	4-37
4.9.3	LPA11-K QIO Functions Program (Program C) _____	4-43

CHAPTER 5 LINE PRINTER DRIVER 5-1

5.1	SUPPORTED LINE PRINTER DEVICES _____	5-1
5.1.1	LP11 Line Printer Controller _____	5-1
5.1.2	DMF32 and DMB32 Line Printer Controllers _____	5-1
5.1.3	LP27 Line Printer _____	5-1
5.1.4	LA11 DECprinter I _____	5-2
5.1.5	LN01 Laser Page Printer _____	5-2

Contents

5.1.6	LN03 Laser Page Printer	5-2
<hr/>		
5.2	DRIVER FEATURES	5-2
5.2.1	Output Character Formatting	5-2
5.2.2	Error Recovery	5-3
<hr/>		
5.3	DEVICE INFORMATION	5-3
<hr/>		
5.4	LINE PRINTER FUNCTION CODES	5-5
5.4.1	Write	5-5
5.4.1.1	Write Function Carriage Control • 5-6	
5.4.2	Sense Printer Mode	5-8
5.4.3	Set Mode	5-9
<hr/>		
5.5	I/O STATUS BLOCK	5-10
<hr/>		
5.6	PROGRAMMING EXAMPLE	5-10

CHAPTER 6 MAGNETIC TAPE DRIVERS

6-1

6.1	SUPPORTED MAGNETIC TAPE CONTROLLERS	6-2
6.1.1	TM03 Magnetic Tape Controller	6-2
6.1.2	TS11 Magnetic Tape Controller	6-2
6.1.3	TM78 Magnetic Tape Controller	6-2
6.1.4	TU80 Magnetic Tape Subsystem	6-3
6.1.5	TU81 and TA81 Magnetic Tape Subsystems	6-3
6.1.6	TK50 Cartridge Tape System	6-3
<hr/>		
6.2	DRIVER FEATURES	6-3
6.2.1	Master Adapters and Slave Formatters	6-4
6.2.2	Data Check	6-4
6.2.3	Error Recovery	6-5
6.2.4	Streaming Tape Systems	6-5
<hr/>		
6.3	DEVICE INFORMATION	6-6
<hr/>		
6.4	MAGNETIC TAPE FUNCTION CODES	6-8
6.4.1	Read	6-12
6.4.2	Write	6-13

6.4.3	Rewind _____	6-14
6.4.4	Skip File _____	6-14
6.4.5	Skip Record _____	6-15
6.4.5.1	Logical End-of-Volume Detection • 6-15	
6.4.6	Write End-of-File _____	6-16
6.4.7	Rewind Offline _____	6-16
6.4.8	Unload _____	6-16
6.4.9	Sense Tape Mode _____	6-16
6.4.10	Set Mode _____	6-17
6.4.11	Data Security Erase _____	6-21
6.4.12	Pack Acknowledge _____	6-21
6.4.13	Available _____	6-21
<hr/>		
6.5	I/O STATUS BLOCK	6-21
<hr/>		
6.6	PROGRAMMING EXAMPLES	6-22
<hr/>		
CHAPTER 7 MAILBOX DRIVER		7-1
<hr/>		
7.1	MAILBOX OPERATIONS	7-1
7.1.1	Creating Mailboxes _____	7-1
7.1.2	Deleting Mailboxes _____	7-2
7.1.3	Mailbox Message Format _____	7-3
7.1.4	Mailbox Protection _____	7-3
<hr/>		
7.2	DEVICE INFORMATION	7-4
<hr/>		
7.3	MAILBOX FUNCTION CODES	7-5
7.3.1	Read _____	7-6
7.3.2	Write _____	7-7
7.3.3	Write End-of-File Message _____	7-8
7.3.4	Set Attention AST _____	7-9
7.3.5	Set Protection _____	7-11
<hr/>		
7.4	I/O STATUS BLOCK	7-12
<hr/>		
7.5	PROGRAMMING EXAMPLE	7-13

Contents

CHAPTER 8	TERMINAL DRIVER	8-1
<hr/>		
8.1	SUPPORTED TERMINAL DEVICES	8-1
<hr/>		
8.2	TERMINAL DRIVER FEATURES	8-2
8.2.1	Input Processing	8-3
8.2.1.1	Command Line Editing and Command Recall • 8-3	
8.2.1.2	Control Characters and Special Keys • 8-4	
8.2.1.3	Read Verify • 8-7	
8.2.1.4	Escape and Control Sequences • 8-8	
8.2.1.5	Type-Ahead Feature • 8-9	
8.2.1.6	Line Terminators • 8-10	
8.2.1.7	Special Operating Modes • 8-10	
8.2.2	Output Processing	8-11
8.2.2.1	Duplex Modes • 8-11	
8.2.2.2	Formatting of Output • 8-12	
8.2.3	Dial-Up Support	8-12
8.2.3.1	Modem Signal Control • 8-12	
8.2.3.2	Hangup on Logging Out • 8-16	
8.2.3.3	Preservation of a Process Across Hangups • 8-16	
8.2.4	Terminal/Mailbox Interaction	8-17
8.2.5	Autobaud Detection	8-18
8.2.6	Out-of-Band Control Character Handling	8-19
<hr/>		
8.3	DEVICE INFORMATION	8-19
8.3.1	Terminal Characteristics Categories	8-26
<hr/>		
8.4	TERMINAL FUNCTION CODES	8-27
8.4.1	Read	8-27
8.4.1.1	Function Modifier Codes for Read QIO Functions • 8-28	
8.4.1.2	Read Function Terminators • 8-29	
8.4.1.3	Itemlist Read Operations • 8-30	
8.4.1.4	Read Verify Function • 8-35	
8.4.2	Write	8-37
8.4.2.1	Function Modifier Codes for Write QIO Functions • 8-37	
8.4.2.2	Write Function Carriage Control • 8-38	
8.4.3	Set Mode	8-40
8.4.3.1	Hangup Function Modifier • 8-44	
8.4.3.2	Enable CTRL/C AST and Enable CTRL/Y AST Function Modifiers • 8-44	
8.4.3.3	Set Modem Function Modifier • 8-45	
8.4.3.4	Loopback Function Modifier • 8-46	
8.4.3.5	Enable Out-of-Band AST Function Modifier • 8-47	
8.4.3.6	Broadcast Function Modifier • 8-48	
8.4.4	LAT Port Driver QIO Interface	8-49

8.4.4.1	LAT Port Driver Functions • 8-50	
8.4.4.2	Application Services Creation • 8-52	
8.4.4.3	Hangup Notification • 8-53	
8.4.5	Sense Mode and Sense Characteristics _____	8-53
8.4.5.1	Type-ahead Count Function Modifier • 8-54	
8.4.5.2	Read Modem Function Modifier • 8-55	
8.4.5.3	Broadcast Function Modifier • 8-56	
<hr/>		
8.5	I/O STATUS BLOCK	8-56
<hr/>		
8.6	PROGRAMMING EXAMPLES	8-59
<hr/>		
APPENDIX A	I/O FUNCTION CODES	A-1
<hr/>		
A.1	ACP-QIO INTERFACE DRIVER	A-1
<hr/>		
A.2	CARD READER DRIVER	A-2
<hr/>		
A.3	DISK DRIVERS	A-2
<hr/>		
A.4	LABORATORY PERIPHERAL ACCELERATOR DRIVER	A-4
<hr/>		
A.5	LINE PRINTER DRIVER	A-5
<hr/>		
A.6	MAGNETIC TAPE DRIVERS	A-6
<hr/>		
A.7	MAILBOX DRIVER	A-7
<hr/>		
A.8	TERMINAL DRIVER	A-8
<hr/>		
APPENDIX B	TABLES	B-1
<hr/>		
B.1	DEC MULTINATIONAL CHARACTER SET	B-1
<hr/>		
B.2	TERMINAL SEQUENCES AND MODES	B-9

Contents

INDEX

EXAMPLES

3-1	Disk Program Example _____	3-28
4-1	LPA11-K High-Level Language Program (Program A) ____	4-36
4-2	LPA11-K High-Level Language Program (Program B) ____	4-38
4-3	LPA11-K QIO Functions Program (Program C) _____	4-44
5-1	Line Printer Program Example _____	5-11
6-1	Magnetic Tape Program Example _____	6-23
6-2	Device Characteristic Program Example _____	6-27
6-3	Set Mode and Sense Mode Program Example _____	6-28
7-1	Mailbox Driver Program Example _____	7-14
8-1	Terminal Program Example _____	8-60
8-2	Read Verify Program Example _____	8-69
8-3	LAT Application Device Program _____	8-73

FIGURES

1-1	ACP-QIO Interface _____	1-1
1-2	ACP Device- or Function-Dependent Arguments _____	1-3
1-3	ACP Device/Function Argument Descriptor Format _____	1-3
1-4	File Information Block Format _____	1-4
1-5	Typical Short File Information Block _____	1-5
1-6	Attribute Control Block Format _____	1-17
1-7	ACP-QIO Record Attributes Area _____	1-21
1-8	ACP-QIO Attributes Statistics Block _____	1-23
1-9	Quota File Transfer Block _____	1-39
1-10	IOSB Contents - ACP-QIO Functions _____	1-40
2-1	A Card Reader Batch Job _____	2-3
2-2	Binary and Packed Column Storage _____	2-7
2-3	Set Mode Characteristics Buffer _____	2-8
2-4	Set Characteristic Buffer _____	2-11
2-5	IOSB Contents _____	2-11
3-1	Disk Physical Address _____	3-6
3-2	Dual-Ported Disk Drives _____	3-8
3-3	Starting Physical Address _____	3-21
3-4	Physical Cylinder Number Format _____	3-22
3-5	IOSB Contents _____	3-27

3-6	IOSB Contents - Sense Mode _____	3-27
4-1	Relationship of Supporting Software to LPA11-K _____	4-4
4-2	Data Transfer Command Table _____	4-11
4-3	Buffer Queue Control _____	4-15
4-4	I/O Functions IOSB Content _____	4-32
5-1	P4 Carriage Control Specifier _____	5-6
5-2	Write Function Carriage Control (Prefix and Postfix Coding) _____	5-8
5-3	Set Mode Buffer _____	5-9
5-4	Set Characteristics Buffer _____	5-9
5-5	IOSB Contents — Write Function _____	5-10
5-6	IOSB Contents — Set Mode Function _____	5-10
6-1	IO\$_SKIPFILE Argument _____	6-14
6-2	IO\$_SKIPRECORD Argument _____	6-15
6-3	Sense Mode P1 Buffer _____	6-17
6-4	Set Mode Characteristics Buffer _____	6-18
6-5	Set Characteristics Buffer _____	6-19
6-6	IOSB Contents _____	6-22
7-1	Multiple Mailbox Channels _____	7-3
7-2	Typical Mailbox Message Format _____	7-4
7-3	Read Mailbox _____	7-7
7-4	Write Mailbox _____	7-8
7-5	Write Attention AST (Read Unsolicited Data) _____	7-10
7-6	Read Attention AST _____	7-10
7-7	Protection Mask _____	7-11
7-8	IOSB Contents - Read Function _____	7-12
7-9	IOSB Contents - Write Function _____	7-12
7-10	IOSB Contents - Set Protection Function _____	7-13
8-1	Modem Control - Two-Way Simultaneous Operation _____	8-14
8-2	Terminal Mailbox Message Format _____	8-18
8-3	Short and Long Forms of Terminator Mask Quadwords _____	8-30
8-4	Itemlist Read Descriptor _____	8-31
8-5	P4 Carriage Control Specifier _____	8-38
8-6	Write Function Carriage Control (Prefix and Postfix Coding) _____	8-41
8-7	Set Mode and Set Characteristics Buffers _____	8-42
8-8	Set Mode P1 Block _____	8-45
8-9	Relationship of Out-of-Band Function with Control Characters _____	8-48
8-10	IO\$_LT_MAP_PORT Item List _____	8-52
8-11	Sense Mode Characteristics Buffer _____	8-54

Contents

8-12	Sense Mode Characteristics Buffer (type-ahead) _____	8-55
8-13	Sense Mode P1 Block _____	8-55
8-14	IOSB Contents—Read Function _____	8-57
8-15	IOSB Contents—Itemlist Read Function _____	8-57
8-16	IOSB Contents—Write Function _____	8-57
8-17	IOSB Contents—Set Mode, Set Characteristics, Sense Mode, and Sense Characteristics Functions _____	8-58
8-18	IOSB Contents—LAT Port Driver Function _____	8-59

TABLES

1-1	Contents of the File Information Block _____	1-5
1-2	FIB Fields (Lookup Control) _____	1-9
1-3	FIB Fields (Access Control) _____	1-11
1-4	FIB Fields (Extend Control) _____	1-13
1-5	FIB Fields (Truncate Control) _____	1-16
1-6	Attribute Control Block Fields _____	1-17
1-7	ACP—QIO Attributes _____	1-18
1-8	File Characteristics Bits _____	1-21
1-9	ACP Record Attributes Values _____	1-22
1-10	Contents of the Statistics Block _____	1-24
1-11	Disk Quota Functions (Enable/Disable) _____	1-37
1-12	Disk Quota Functions (Individual Entries) _____	1-38
2-1	Card Reader Device-Independent Characteristics _____	2-5
2-2	Device-Dependent Characteristics for Card Readers _____	2-5
2-3	Card Reader I/O Functions _____	2-6
2-4	Set Mode and Set Characteristic Card Reader Characteristics _____	2-8
2-5	Card Reader Codes _____	2-8
3-1	Supported Disk Devices _____	3-1
3-2	Disk Device Characteristics _____	3-16
3-3	Disk I/O Functions _____	3-18
4-1	Minimum and Maximum Configurations per LPA11-K _____	4-1
4-2	LPA11-K Device-Independent Characteristics _____	4-5
4-3	LPA11-K Device-Dependent Characteristics _____	4-5
4-4	VAX Procedures for the LPA11-K _____	4-13
4-5	Subroutine Argument Usage _____	4-15
4-6	LPA\$IGTBUF Call — IBUFNO and IOSB Contents _____	4-28
4-7	LPA\$IWTBUF Call — IBUFNO and IOSB Contents _____	4-29
4-8	Program A Variables _____	4-36

4-9	Program B Variables _____	4-38
5-1	Printer Device-Independent Characteristics _____	5-4
5-2	Device-Dependent Characteristics for Line Printers _____	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) _____	5-7
6-1	Supported Magnetic Tape Devices _____	6-1
6-2	Magnetic Tape Device-Independent Characteristics _____	6-6
6-3	Device-Dependent Information for Tape Devices _____	6-7
6-4	Extended Device Characteristics for Tape Devices _____	6-8
6-5	Magnetic Tape I/O Functions _____	6-9
6-6	Set Mode and Set Characteristics Magnetic Tape Characteristics _____	6-19
6-7	Extended Device Characteristics for Tape Devices _____	6-20
7-1	Mailbox Read and Write Operations _____	7-1
7-2	Mailbox Characteristics _____	7-5
8-1	Supported Terminal Devices _____	8-1
8-2	Terminal Control Characters _____	8-4
8-3	Control and Data Signals (Full Modem Mode Configuration) _____	8-15
8-4	Terminal Device-Independent Characteristics _____	8-20
8-5	Terminal Characteristics _____	8-20
8-6	Extended Terminal Characteristics _____	8-22
8-7	Read QIO Function Modifiers for the Terminal Driver _____	8-28
8-8	Item Codes for Itemlist Read Operations for the Terminal Driver _____	8-31
8-9	Write QIO Function Modifiers for the Terminal Driver _____	8-37
8-10	Write Function Carriage Control (FORTRAN: byte 0 not equal to 0) _____	8-39
8-11	Write Function Carriage Control (P4 byte 0 = 0) _____	8-40
8-12	Broadcast Requester IDs _____	8-49
8-13	IO\$M_LT_CONNECT Request Status _____	8-51
8-14	IO\$M_LT_MAP_PORT and IO\$M_LT_RATING Request Status _____	8-52
8-15	LAT Rejection Codes _____	8-59
B-1	DEC Multinational Character Set _____	B-1
B-2	Sequences and Modes _____	B-10

Preface

Intended Audience

This manual is intended for system programmers who want to take advantage of the time and space savings that result from direct use of I/O devices. Users of VMS who do not require such detailed knowledge of I/O drivers can use the device-independent services described in the *VMS Record Management Services Manual*.

Document Structure

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

- Chapter 1 describes the Queue I/O (QIO) interface to file system ancillary control processes (ACPs).
- Chapters 2 through 8 describe the use of VMS file-structured and real-time I/O device drivers, the drivers for storage devices such as disks and magnetic tapes, and terminal devices supported by VMS:
 - Chapter 2 discusses the card reader driver.
 - Chapter 3 discusses disk drivers.
 - Chapter 4 discusses the LPA11-K driver.
 - Chapter 5 discusses the line printer drivers.
 - Chapter 6 discusses the magnetic tape drivers.
 - Chapter 7 discusses the mailbox driver.
 - Chapter 8 discusses the terminal driver.
- Appendix A summarizes the QIO function codes, arguments, and function modifiers used by the drivers listed above.
- Appendix B lists the DEC Multinational Character Set and the ANSI and DIGITAL-private escape sequences for terminals.

Associated Documents

The following documents provide additional information:

- *VMS System Services Reference Manual*
- *VMS Software Information Management Handbook*
- *VMS Software VMS System Software Handbook*
- *Guide to VMS Programming Resources*
- *VMS Record Management Services Manual*
- *LPA11-K Laboratory Peripheral Accelerator User's Guide*

Preface

- *VMS Networking Manual*
- *VMS System Messages and Recovery Procedures Reference Volume*
- *VMS Device Support Manual*

Conventions

Convention	Meaning
<code>RET</code>	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.)
<code>CTRL/C</code>	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
<code>\$</code> <code>05-JUN-1988 11:55:22</code>	In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red.
<code>\$</code> <code>.</code> <code>.</code> <code>.</code>	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.
<code>input-file, . . .</code>	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
<code>[logical-name]</code>	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)

quotation marks
apostrophes

The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

-

Hyphens in coding examples indicate that additional arguments to the request are provided on the line that follows. For example:

```
CMDOFAB: $FAB fac=put,fnm=sys$output,-  
          mrs=132,rat=cr,rfm=var  
CMDORAB: $RAB ubf=cmdbuf,usz=cmdbsz,-  
          fab=cmdofab
```

numbers

Unless otherwise noted, all numbers in the text are assumed to be decimal. Nondecimal radices—binary, octal, or hexadecimal—are explicitly indicated in the coding examples.

New and Changed Features

This revision of the *VMS I/O User's Reference Manual: Part I* reflects the technical changes since VMS Version 4.4. The following chapters contain new or changed information:

- Chapter 1
 - Sections 1.3.1.3 and 1.6.1.3 describe directory entry protection and creation.
- Chapter 2
 - Section 2.2.1.2 describes the methods you use to change the translation mode.
 - Section 2.2.2 describes how you submit batch jobs through the card reader.
- Chapter 3
 - Table 3-1 includes the new disk devices supported for V5.0.
 - Section 3.2.1.2.1 describes a restriction on dual-ported non-DSA disks in a VAXcluster.
 - Section 3.2.7 provides information on the VAXstation 2000/MicroVAX 2000 disk driver.
- Chapter 6
 - Section 6.4 provides an example of how to correctly define the P1 parameter in a IO\$_SKIPRECORD QIO.
 - Table 6-1 includes the new magnetic tape devices supported for V5.0.
- Chapter 8
 - Section 8.4.3 describes the consequences if you use TT2\$V_FALLBACK for a disconnected virtual terminal (_VTax:) or if the Terminal Fallback Facility is not activated.
 - Section 8.4.4 describes the LAT port driver QIO interface.

1

ACP—QIO Interface

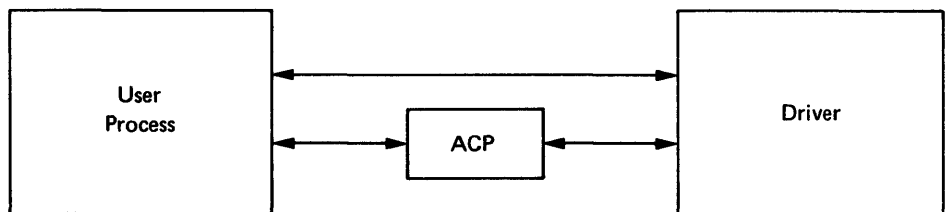
An ancillary control process (ACP) is a process that interfaces between the user process and the driver, and performs functions that supplement the driver's functions. Virtual I/O operations involving file-structured devices (disks and magnetic tapes) often require ACP intervention. In most cases, ACP intervention is requested by VMS 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 1-1.

Executing physical and logical I/O operations on a device being managed by a file ACP will interfere with the operation of the ACP and will result in unpredictable consequences, including system failure in certain cases.

In addition to the ACP, the VMS operating system also provides the XQP (extended QIO processor) facility to supplement the QIO driver's functions when performing virtual I/O operations on file-structured devices (ACP for Files-11 On-Disk Structure Level 1 and XQP for Files-11 On-Disk Structure Level 2). However, rather than being a separate process, the XQP executes as a kernel mode thread in the process of its caller.

This chapter describes the QIO interface to ACPs for disk and magnetic tape devices (file system ACPs). The sample program in Chapter 6 performs QIO operations to the magnetic tape ACP.

Figure 1-1 ACP—QIO Interface



ZK-635-82

This section also describes a number of structures and field names of the form xxx\$name. A VAX MACRO program can define symbols of this form by invoking the \$xxxDEF macro.

The following macros are available in SYS\$LIBRARY:STARLET.MLB:

- \$IODEF
- \$FIBDEF
- \$ATRDEF
- \$SBKDEF

ACP—QIO Interface

The following macros are available in SYS\$LIBRARY:LIB.MLB:

```
$FATDEF  
$DQFDEF  
$FCHDEF
```

Programs written in BLISS-32 can use these symbols by referencing them and including the correct library, SYS\$LIBRARY:STARLET.L32 (for the macros listed under SYS\$LIBRARY:STARLET.MLB), and SYS\$LIBRARY:LIB.L32 (for the macros listed under SYS\$LIBRARY:LIB.MLB).

References to ANSI refer to the *American National Standard Magnetic Tape Labels and File Structures for Information Interchange, ANSI X3.27-1978*.

1.1 ACP Functions and Encoding

All VMS ACP functions can be expressed using seven function codes and four function modifiers. The function codes are as follows:

- IO\$_CREATE—Creates a directory entry or file
- IO\$_ACCESS—Searches a directory for a specified file and accesses the file, if found
- IO\$_DEACCESS—Deaccesses a file and, if specified, writes the final attributes in the file header
- IO\$_MODIFY—Modifies the file attributes and file allocation
- IO\$_DELETE—Deletes a directory entry and file header
- IO\$_MOUNT—Informs the ACP when a volume is mounted; requires MOUNT privilege
- IO\$_ACPCONTROL—Performs miscellaneous control functions

The function modifiers are:

- IO\$_M_ACCESS—Opens a file on the user's channel
- IO\$_M_CREATE—Creates a file
- IO\$_M_DELETE—Deletes a file (or marks it for deletion)
- IO\$_M_DMOUNT—Dismounts a volume

In addition to the function codes and modifiers, VMS ACPs take five device- or function-dependent arguments, as shown in Figure 1-2. The first argument, P1, is the address of the file information block (FIB) descriptor. Section 1.2 describes the FIB in detail.

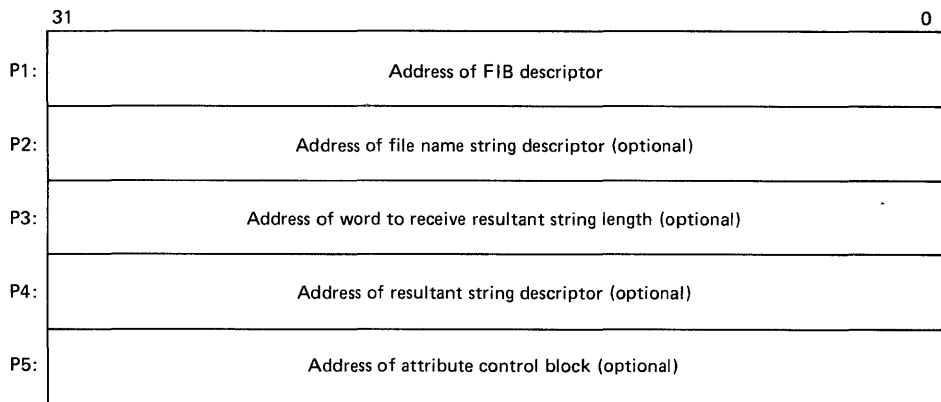
The second argument, P2, is an optional argument used in directory operations. It specifies the address of the descriptor for the file name string to be entered in the directory.

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 of these arguments are optional.

ACP—QIO Interface

1.1 ACP Functions and Encoding

Figure 1–2 ACP Device- or Function-Dependent Arguments



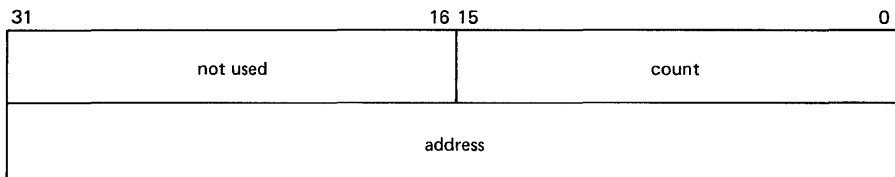
ZK-636-82

The fifth argument, P5, is an optional argument containing the address of the attribute control block. Section 1.3.5 describes the attribute control block in detail.

All areas of memory specified by the descriptors must be capable of being read or written to.

Figure 1–3 shows the format for the descriptors. The count field is the length in bytes of the item described.

Figure 1–3 ACP Device/Function Argument Descriptor Format



ZK-637-82

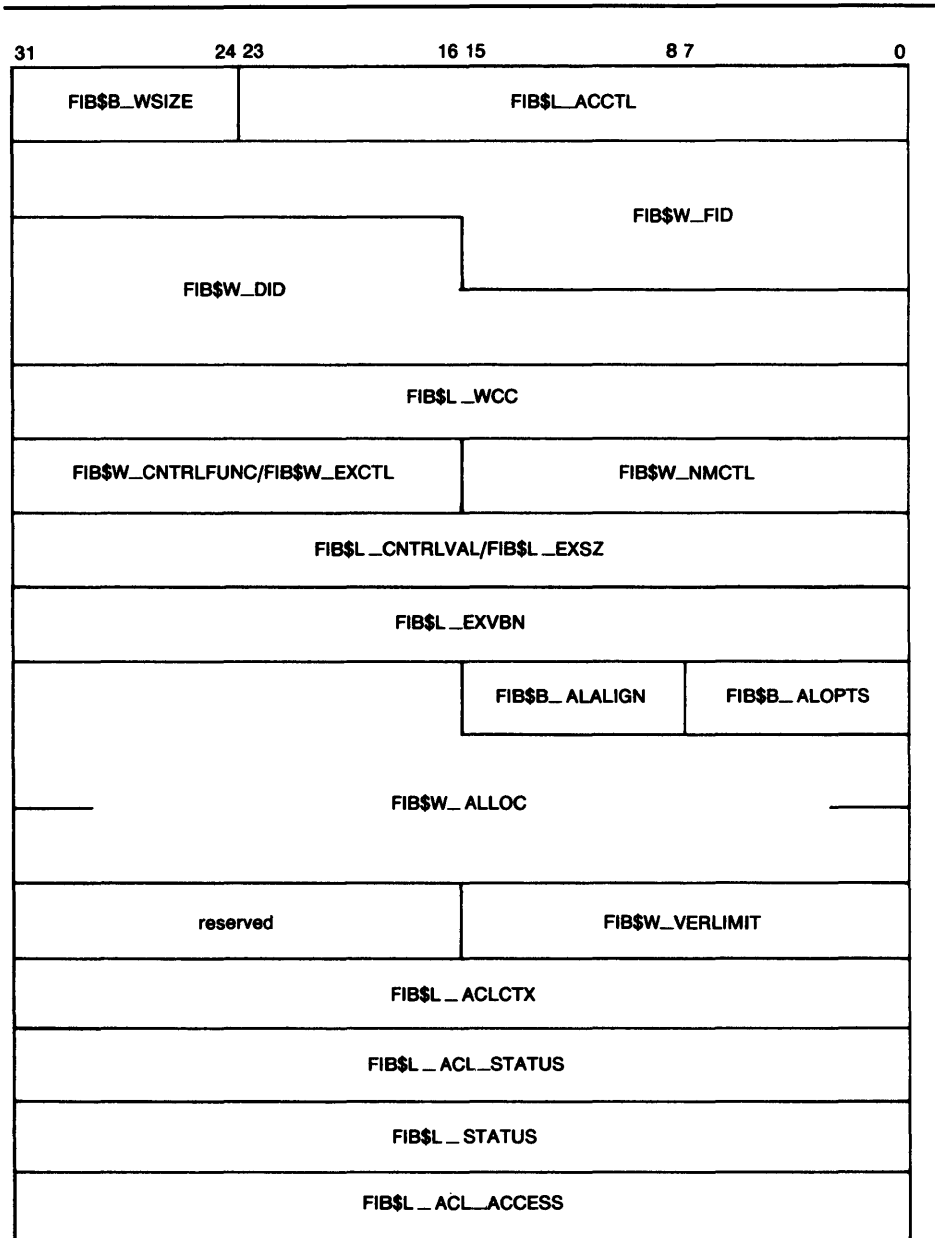
1.2 File Information Block (FIB)

The file information block (FIB) contains much of the information that is exchanged between the user process and the ACP. Figure 1–4 shows the format of the FIB. The FIB must be writable. Because the FIB is passed by a descriptor (see Figure 1–3), its length can vary. Thus, a short FIB can be used in ACP calls that do not need arguments near the end of the FIB. The ACP treats the omitted portion of the FIB as if it were 0. Figure 1–5 shows the format of a typical short FIB that would be used to open an existing file. Table 1–1 gives a brief description of each of the FIB fields. More detailed descriptions are provided in Sections 1.3 and 1.6.

ACP—QIO Interface

1.2 File Information Block (FIB)

Figure 1—4 File Information Block Format



ZK-638-82

ACP—QIO Interface

1.2 File Information Block (FIB)

Figure 1–5 Typical Short File Information Block

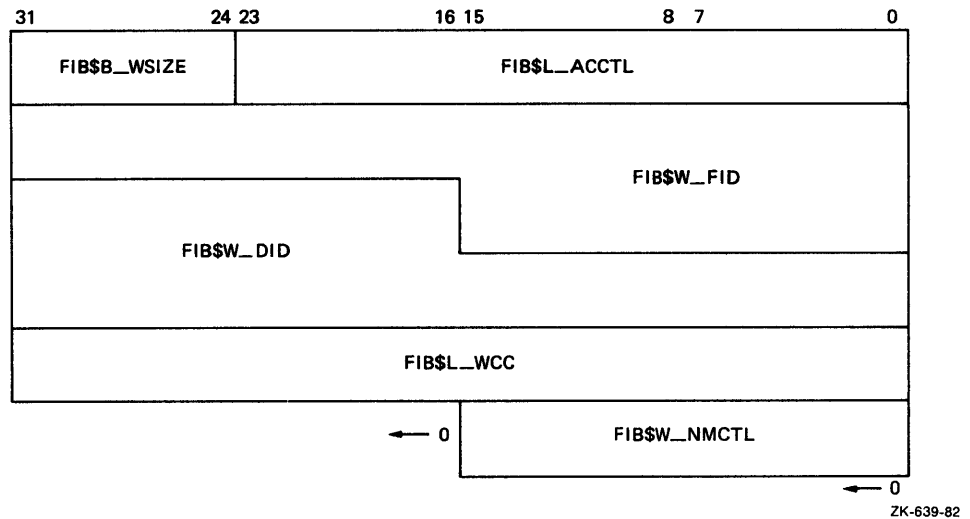


Table 1–1 Contents of the File Information Block

Field	Subfields	Meaning
FIB\$L_ACCTL		Contains flag bits that control the access to the file. Sections 1.3.1.1, 1.3.2.1, 1.6.1.1, 1.6.4.1, and 1.6.5 describe the FIB\$L_ACCTL field flag bits.
FIB\$B_WSIZE		Controls the size of the file window used to map a disk file. If a window size of 255 is specified, the file is mapped completely through the use of segmented windows.
FIB\$W_FID		Specifies the file identification. You supply 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. A 0 file identifier can be specified when an operation is performed on a file that is already open on a particular channel. The ACP returns the file identifier of the open file. The following subfields are defined:
	FIB\$W_FID_NUM	File number.

ACP—QIO Interface

1.2 File Information Block (FIB)

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

Field	Subfields	Meaning
	FIB\$W_FID_SEQ	File sequence number.
	FIB\$W_FID_RVN	Relative volume number (only for magnetic tape devices).
	FIB\$B_FID_RVN	Relative volume number (only for disk devices).
	FIB\$B_FID_NMX	File number extension (only for disk devices).
FIB\$W_DID		Contains the file identifier of the directory file. The following subfields are defined:
	FIB\$W_DID_NUM	File number.
	FIB\$W_DID_SEQ	File sequence number.
	FIB\$W_DID_RVN	Relative volume number (only for magnetic tape devices).
	FIB\$B_DID_RVN	Relative volume number (only for disk devices).
	FIB\$B_DID_NMX	File number extension (only for disk devices).
FIB\$L_WCC		Maintains position context when processing wildcard directory operations.
FIB\$W_NMCTL		Contains flag bits that control the processing of a name string in a directory operation. Sections 1.3.1.1 and 1.6.1.1 describe the FIB\$W_NMCTL field flag bits.
FIB\$W_EXCTL		Contains flag bits that specify extend control for disk devices. Sections 1.3.3.1 and 1.3.4.1 describe the FIB\$W_EXCTL field flag bits.
FIB\$W_CNTRLFUNC		In an IO\$_ACPCONTROL function, this field contains the code that specifies which ACP control function is to be performed (see Section 1.6.7). This field overlays FIB\$W_EXCTL.
	FIB\$C_USEREOT	User EOT mode. In an IO\$_CREATE or IO\$_ACCESS function, you can set this mode on a per file basis. (See Sections 1.6.1 and 1.6.2.)

ACP—QIO Interface

1.2 File Information Block (FIB)

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

Field	Subfields	Meaning
FIB\$_EXSZ		Specifies the number of blocks to be allocated in an extend operation on a disk file.
FIB\$_CNTRLVAL		Contains a control function value used in an IO\$_ACPCONTROL function (see Section 1.6.7). The interpretation of the value depends on the control function specified in FIB\$_CNTRLFUNC. This field overlays FIB\$_EXSZ.
FIB\$_EXVBN		Specifies the starting disk file virtual block number at which a file is to be truncated.
FIB\$_ALOPTS		Contains option bits that control the placement of allocated blocks. Section 1.3.3.1 describes the FIB\$_ALOPTS field flag bits.
FIB\$_ALALIGN		Contains the interpretation mode of the allocation (FIB\$_ALLOC) field.
FIB\$_ALLOC		Contains the desired physical location of the blocks being allocated. Interpretation of the field is controlled by the FIB\$_ALALIGN field. The following subfields are defined:
	FIB\$_LOC_FID	Three-word related file ID for RFI placement.
	FIB\$_LOC_NUM	Related file number.
	FIB\$_LOC_SEQ	Related file sequence number.
	FIB\$_LOC_RVN	Related file RVN or placement RVN.
	FIB\$_LOC_NMX	Related file number extension.
	FIB\$_LOC_ADDR	Placement LBN, cylinder, or VBN.
FIB\$_VERLIMIT		Contains the version limit of the directory entry.
FIB\$_ACLCTX		Maintains position context when processing ACL attributes from the attribute (P5) list.

ACP—QIO Interface

1.2 File Information Block (FIB)

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

Field	Subfields	Meaning
FIB\$_ACL_STATUS		Status of the requested ACL attribute operation, if any. The ACL attributes are included in Table 1–7. If no ACL attributes are given, SS\$_NORMAL is returned here.
FIB\$_STATUS		Access status. Applies to all major functions. The following bits are supported:
	FIB\$_ALT_REQ	Set to indicate whether the alternate access bit is required for the current operation. If not set, the alternate access bit is optional.
	FIB\$_ALT_GRANTED	If FIB\$_ALT_REQ = 0, the FIB bit returned from the file system is set if the alternate access check succeeded.
FIB\$_ALT_ACCESS		A 32-bit mask that represents an access mask to check against file protection; for example, opens a file for read access and checks whether it can be deleted. The mask has the same configuration as the standard protection mask.

1.3 ACP Subfunctions

The operations that the ACP performs can be organized into two categories: major ACP functions and subfunctions. Each ACP operation performs one major function. That function is specified by an I/O function code, such as IO\$_ACCESS, IO\$_CREATE, or IO\$_MODIFY. While executing the major function, one or more subfunctions can be performed. A subfunction is an operation such as looking up, accessing, or extending a file. Most subfunctions can be executed by more than one of the major functions. Sections 1.3.1 through 1.3.5 describe the following subfunctions in detail:

- Directory Lookup
- Access
- Extend
- Truncate
- Read Attributes
- Write Attributes

Section 1.6, which contains the descriptions of the major functions, lists the subfunctions available to each major function.

1.3.1 Directory Lookup

The directory lookup subfunction is used to search for a file in a disk directory or on a magnetic tape. This subfunction can be invoked using the major functions IO\$_ACCESS, IO\$_MODIFY, IO\$_DELETE, and IO\$_ACPCONTROL. A directory lookup occurs if the directory file ID field in the FIB (FIB\$_DID) is a nonzero number.

1.3.1.1 Input Parameters

Table 1–2 lists the FIB fields that control the processing of a lookup subfunction.

Table 1–2 FIB Fields (Lookup Control)

Field	Field Values	Meaning
FIB\$_NMCTL		Name string control. The following name control bits are applicable to a lookup operation:
	FIB\$_WILD	Set if name string contains wildcards. Setting this bit causes wildcard context to be returned in FIB\$_WCC.
	FIB\$_ALLNAM	Set to match all name field values.
	FIB\$_ALLTYP	Set to match all field type values.
	FIB\$_ALLVER	Set to match all version field values.
	FIB\$_FINDFID	Set to search a directory for the file identifier in FIB\$_FID.
FIB\$_FID		File identification. The file ID of the file found is returned in this field.
FIB\$_DID		Contains the file identifier of the directory file. This field must be a nonzero number.
FIB\$_WCC		Maintains position context when processing wildcard directory operations.
FIB\$_ACCTL		The following access control flag is applicable to a lookup subfunction:
	FIB\$_REWIND	Set to rewind magnetic tape before lookup. If not set, a magnetic tape is searched from its current position.

QIO arguments P2 through P6 are passed as values. The second argument, P2, specifies the address of the descriptor for the file name string to be searched for in the directory.

The file name string must have one of the following two formats:

name.type;version

name.type.version

ACP—QIO Interface

1.3 ACP Subfunctions

The name and type can be any combination of alphanumeric characters, and the dollar sign (\$), asterisk (*), and percent (%) characters. The version must consist of numeric characters optionally preceded by a minus sign (-) (only for disk devices) or a single asterisk. The total number of alphanumeric and percent characters in the name field and in the type field must not exceed 39. Any number of additional asterisks can be present.

If any of the bits `FIB$M_ALLNAM`, `FIB$M_ALLTYP`, and `FIB$M_ALLVER` are set, then the contents of the corresponding field in the name string are ignored and the contents are assumed to be an asterisk.

Note that the file name string cannot contain a directory string. The directory is specified by the `FIB$W_DID` field (see Table 1-1). Only VMS RMS can process directory strings.

Argument P3 is the address of a word to receive the resultant file name string length.

Argument P4 is the address of a descriptor for a buffer to receive the resultant file name string. The resultant string is not padded. The P3 and P4 arguments are optional.

1.3.1.2 Operation

The system searches either the directory file specified by `FIB$W_DID` or the magnetic tape for the file name specified in the P2 file name parameter. The actual file name found and its length are returned in the P3 and P4 length and result string buffers. The file ID of the file found is returned in `FIB$W_FID` and can be used in subsequent operations as the major function is processed.

Zero and negative version numbers have special significance in a disk lookup operation. Specifying 0 as a version number causes the latest version of the file to be found. Specifying -1 locates the second most recent version, -2 the third most recent, and so forth. Specifying a version of -0 locates the lowest numbered version of the file. For magnetic tape lookups, a version number of 0 locates the first occurrence of the file encountered; negative version numbers are not allowed.

Wildcard lookups are performed by specifying the appropriate wildcard characters in the name string and setting `FIB$M_WILD`. (The name control bits `FIB$M_ALLNAM`, `FIB$M_ALLTYP`, and `FIB$M_ALLVER` can also be used in searching for wildcard entries, but they are intended primarily for compatibility mode use.) On the first lookup, `FIB$L_WCC` should contain zero entries. On each lookup, the ACP returns a nonzero value in `FIB$L_WCC`, which must be passed back on the next lookup call. In addition, you must pass the resultant name string returned by the previous lookup using the P4 result string buffer, and its length in the P3 result length word. This string is used together with `FIB$L_WCC` to continue the wildcard search at the correct position in the directory.

Perform a lookup by file ID by setting the name control bit `FIB$M_FINDFID`. When this bit is set, the system searches the directory for an entry containing the file ID specified in `FIB$W_FID`, and the name of the entry found is returned in the P3 and P4 result parameters. Note that if a directory contains multiple entries with the same file ID, only the first entry can be located with this technique.

Lookups by file ID should be done only when the file name is not available, because lookups by this method are often significantly slower than lookups by file name.

1.3.1.3 Directory Entry Protection

A directory entry is protected with the same protection code as the file itself. For example, if a file is protected against delete access, then the file name has the same protection. Consequently, a nonprivileged user (that is, a user who is not the volume owner or a user who does not have SYSPRV) cannot rename a file because renaming a file is essentially the same as deleting the file name. This protection is applied regardless of the protection on the directory file.

Nonprivileged users can neither write directly into a .DIR;1 directory file nor turn off the directory bit in a directory file header.

1.3.2 Access

The access subfunction is used to open a file so that virtual read or write operations can be performed. This subfunction can be invoked using the major functions IO\$_CREATE and IO\$_ACCESS (see Sections 1.6.1 and 1.6.2). An access subfunction is performed if the IO\$_M_ACCESS modifier is specified in the I/O function code.

1.3.2.1 Input Parameters

Table 1–3 lists the FIB fields that control the processing of an access subfunction.

Table 1–3 FIB Fields (Access Control)

Field	Field Values	Meaning
FIB\$_ACCTL		Specifies field values that control access to the file. The following access control bits are applicable to the access subfunction:
	FIB\$_WRITE	Set for write access; clear for read-only access.
	FIB\$_NOREAD	Set to deny read access to others. (You must have write privilege to the file to use this option.)
	FIB\$_NOWRITE	Set to deny write access to others.
	FIB\$_NOTRUNC	Set to prevent the file from being truncated; clear to allow truncation.
	FIB\$_DLOCK	Set to enable deaccess lock (close check). Used only for disk devices.
		Used to flag a file as inconsistent if the program currently modifying the file terminates abnormally. If the program deaccesses the file without performing a write attributes operation, the file is marked as locked and cannot be accessed until it is unlocked.

ACP—QIO Interface

1.3 ACP Subfunctions

Table 1–3 (Cont.) FIB Fields (Access Control)

Field	Field Values	Meaning
	FIB\$_UPDATE	Set to position at start of a magnetic tape file when opening file for write; clear to position at end-of-file.
	FIB\$_READCK	Set to enable read checking of the file. Virtual reads to the file are performed using a data check operation.
	FIB\$_WRITECK	Set to enable write checking of the file. Virtual writes to the file are performed using a data check operation.
	FIB\$_EXECUTE	Set to access the file in execute mode. The protection check is made against the EXECUTE bit instead of the READ bit. Valid only for requests issued from SUPERVISOR, EXEC, or KERNEL mode.
	FIB\$_NOLOCK	Set to override exclusive access to the file, allowing you to access the file when another user has the file open with FIB\$_NOREAD specified. You must have SYSPRV privilege or ownership of the volume to use this option. FIB\$_NOREAD and FIB\$_NOWRITE must be clear for this option to work.
	FIB\$_NORECORD	Set to inhibit recording of the file's expiration date. If not set, the file's expiration date can be modified, depending on the file retention parameters of the volume.
FIB\$_WSIZE		Controls the size of the file window used to map a disk file. The ACP uses the volume default if FIB\$_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. Note that the window is charged to the user's BYTELIM quota.
FIB\$_FID		Specifies the file identification of the file to be accessed.

1.3.2.2 Operation

The file is opened according to the access control specified (see Table 1–3).

1.3.3 Extend

The extend subfunction is used to allocate space to a disk file. This subfunction can be invoked using the major I/O functions IO\$_CREATE and IO\$_MODIFY (see Sections 1.6.1 and 1.6.4). The extend subfunction is performed if the bit FIB\$_EXTEND is set in the extend control word FIB\$_EXCTL.

1.3.3.1 Input Parameters

Table 1–4 lists the FIB fields that control the processing of an extend subfunction.

Table 1–4 FIB Fields (Extend Control)

Field	Field Values	Meaning
FIB\$_EXCTL		Extend control flags. The following flags are applicable to the extend subfunction:
	FIB\$_EXTEND	Set to enable extension.
	FIB\$_NOHDREXT	Set to inhibit generation of extension file headers.
	FIB\$_ALCON	Allocates contiguous space. The extend operation fails if the necessary contiguous space is not available.
	FIB\$_ALCONB	Allocates the maximum amount of contiguous space. If both FIB\$_ALCON and FIB\$_ALCONB are set, a single contiguous area, whose size is the largest available but not greater than the size requested, is allocated.
	FIB\$_FILCON	Marks the file contiguous. This bit can only be set if the file does not have space already allocated to it.
	FIB\$_ALDEF	Allocates the extend size (FIB\$_EXSZ) or the system default, whichever is greater.
FIB\$_EXSZ		Specifies the number of blocks to allocate to the file.

ACP—QIO Interface

1.3 ACP Subfunctions

Table 1–4 (Cont.) FIB Fields (Extend Control)

Field	Field Values	Meaning
		The number of blocks actually allocated for this operation is returned in this longword. More blocks than requested can be allocated to meet cluster boundaries.
FIB\$_EXVBN		Returns the starting virtual block number of the blocks allocated. FIB\$_EXVBN must initially contain 0 blocks.
FIB\$_ALOPTS		Contains option bits that control the placement of allocated blocks. The following bits are defined:
	FIB\$_EXACT	Set to require exact placement; clear to specify approximate placement. If this bit is set and the specified blocks are not available, the extend operation fails.
	FIB\$_ONCYL	Set to locate allocated space within a cylinder. This option functions correctly only when FIB\$_ALCON or FIB\$_ALCONB is specified.
FIB\$_ALALIGN		Contains the interpretation mode of the allocation (FIB\$_ALLOC) field. One of the following values can be specified:
	(zero)	No placement data. The remainder of the allocation field is ignored.
	FIB\$_CYL	Location is specified as a byte relative volume number (RVN) in FIB\$_LOC_RVN and a cylinder number in FIB\$_LOC_ADDR.
	FIB\$_LBN	Location is specified as a byte RVN in FIB\$_LOC_RVN, followed by a longword logical block number (LBN) in FIB\$_LOC_ADDR.
	FIB\$_VBN	Location is specified as a longword virtual block number (VBN) of the file being extended in FIB\$_LOC_ADDR. A 0 VBN or one that fails to map indicates the end of the file.

ACP—QIO Interface

1.3 ACP Subfunctions

Table 1–4 (Cont.) FIB Fields (Extend Control)

Field	Field Values	Meaning
	FIB\$C_RFI	Location is specified as a three-word file ID in FIB\$W_LOC_FID, followed by a longword VBN of that file in FIB\$L_LOC_ADDR. A 0 file ID indicates the file being extended. A 0 VBN or one that fails to map indicates the end of that file.
FIB\$W_ALLOC		Contains the desired physical location of the blocks being allocated. Interpretation of the field is controlled by the FIB\$B_ALALIGN field. The following subfields are defined:
	FIB\$W_LOC_FID	Three-word related file ID for RFI placement.
	FIB\$W_LOC_NUM	Related file number.
	FIB\$W_LOC_SEQ	Related file sequence number.
	FIB\$B_LOC_RVN	Related file RVN or placement RVN.
	FIB\$B_LOC_NMX	Related file number extension.
	FIB\$L_LOC_ADDR	Placement LBN, cylinder, or VBN.

1.3.3.2 Operation

The specified number of blocks are allocated and appended to the file. The virtual block number assigned to the first block allocated is returned in FIB\$L_EXVBN. The actual number of blocks allocated is returned in FIB\$L_EXSZ.

The actual number of blocks allocated is also returned in the second longword of the user's I/O status block. If a contiguous allocation (FIB\$M_ALCON) fails, the size of the largest contiguous space available on the disk is returned in the second longword of the user's I/O status block.

1.3.4 Truncate

The truncate subfunction is used to remove space from a disk file. This subfunction can be invoked by the major I/O functions IO\$_DEACCESS and IO\$_MODIFY (see Sections 1.6.3 and 1.6.4). The truncate subfunction is performed if the bit FIB\$M_TRUNCATE is set in the extend control word FIB\$W_EXCTL.

ACP—QIO Interface

1.3 ACP Subfunctions

1.3.4.1 Input Parameters

Table 1–5 lists the FIB fields that control the processing of a truncate subfunction.

Table 1–5 FIB Fields (Truncate Control)

Field	Field Values	Meaning
FIB\$W_EXCTL		Extend control flags. The following flags are applicable to the truncate subfunction:
	FIB\$M_TRUNC	Must be set to enable truncation.
	FIB\$M_MARKBAD	Set to append the truncated blocks to the bad block file, instead of returning them to the free storage pool. Only one cluster can be deallocated. This is most easily accomplished by specifying the last VBN of the file in FIB\$L_EXVBN. SYSPRV privilege or ownership of the volume is required to deallocate blocks to the bad block file.
FIB\$L_EXSZ		Returns the actual number of blocks deallocated. FIB\$L_EXSZ must initially contain a value of 0.
FIB\$L_EXVBN		Specifies the first virtual block number to be removed from the file. The actual starting virtual block number of the truncate operation is returned in this field.

1.3.4.2 Operation

Blocks are deallocated from the file, starting with the virtual block specified in FIB\$L_EXVBN and continuing through the end of the file. The actual number of blocks deallocated is returned in FIB\$L_EXSZ. The virtual block number of the first block actually deallocated is returned in FIB\$L_EXVBN. Because of cluster round-up, this value might be greater than the value specified. If FIB\$M_MARKBAD is specified, the truncation VBN is rounded down instead of up, and the value returned in FIB\$L_EXVBN might be less than that specified.

The number of blocks by which FIB\$L_EXVBN was rounded up is returned in the second longword of the I/O status block.

The truncate subfunction normally requires exclusive access to the file at run time. This means, for example, that a file cannot be truncated while multiple writers have access to it.

An exception occurs when a truncate subfunction is requested for a write-accessed file that allows other readers. Although the truncate subfunction returns success status in this instance, the actual file truncation (the return of the truncated blocks to free storage) is deferred until the last reader deaccesses the file. If a new writer accesses the file after the truncate subfunction is requested, but before the last deaccess, the deferred truncation is ignored.

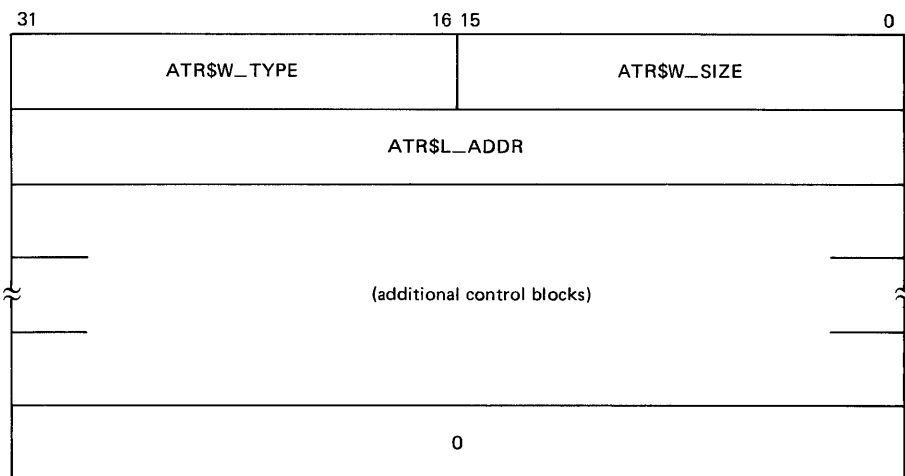
1.3.5 Read/Write Attributes

The read and write attributes subfunctions are used for operations such as reading and writing file protection and creating and revising dates. A read or write attributes operation is invoked by specifying an attribute list with the QIO parameter P5. A read attributes operation can be invoked by the major I/O function IO\$_ACCESS (see Section 1.6.2); a write attributes operation can be invoked by the major I/O functions IO\$_CREATE, IO\$_DEACCESS, and IO\$_MODIFY (see Sections 1.6.1, 1.6.3, and 1.6.4).

1.3.5.1 Input Parameters

The read or write attributes subfunction is controlled by the attribute list specified by P5. The list consists of a variable number of two longword control blocks, terminated by a 0 longword, as shown in Figure 1-6. The maximum number of attribute control blocks in one list is 30. Table 1-6 describes the attribute control block fields.

Figure 1-6 Attribute Control Block Format



ZK-640-82

Table 1-6 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 1-7).
ATR\$W_TYPE	Identifies the individual attribute to be read or written.
ATR\$L_ADDR	Contains the buffer address of the memory space to or from which the attribute is to be transferred. The attribute buffer must be writable.

Table 1-7 lists the valid attributes for ACP-QIO functions. The maximum size (in bytes) is determined by the required attribute configuration. For example, the Radix-50 file name (ATR\$_FILNAM) uses only 6 bytes, but it is always accompanied by the file type and file version, so a total of 10 bytes

ACP—QIO Interface

1.3 ACP Subfunctions

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).

Table 1–7 ACP—QIO Attributes

Attribute Name¹	Maximum Size (bytes)	Meaning
ATR\$C_UCHAR ^{2 4}	4	4-byte file characteristics. (The file characteristics bits are listed following this table.)
ATR\$C_RECATTR ³	32	Record attribute area. Section 1.4 describes the record attribute area in detail.
ATR\$C_FILNAM	10	6-byte Radix–50 file name plus ATR\$C_FILTYP and ATR\$C_FILVER.
ATR\$C_FILTYP	4	2-byte Radix–50 file type plus ATR\$C_FILVER.
ATR\$C_FILVER	2	2-byte binary version number.
ATR\$C_EXPDAT ²	7	Expiration date in ASCII. Format: DDMMYY.
ATR\$C_STATBLK ⁵	32	Statistics block. Section 1.5 describes the statistics block in detail.
ATR\$C_HEADER ⁵	512	Complete file header.
ATR\$C_BLOCKSIZE	2	Magnetic tape block size.
ATR\$C_USERLABEL ⁶	80	User file label.
ATR\$C_ASCDATES ^{2 4}	35	Revision count (2 binary bytes), revision date, creation date, and expiration date, in ASCII. Format: DDMMYY (revision date), HHMMSS (time), DDMMYY (creation date), HHMMSS (time), DDMMYY (expiration date). (The format contains no embedded spaces or commas.)
ATR\$C_ALCONTROL	14	Compatibility mode allocation data.
ATR\$C_ENDLBLAST	4	End of magnetic tape label processing; provides AST control block.

¹Attributes with an ATR\$C_ prefix have two names: one with the ATR\$C_ prefix for the code and one with an ATR\$S_ prefix for the size, which is not included in the list.

²Protected (can be written to only by system or owner).

³Locked (cannot be written to while the file is locked against writers).

⁴Not supported on write operations to MTAACP; defaults are returned on read operations.

⁵Read only.

⁶Not supported for disk devices.

ACP—QIO Interface

1.3 ACP Subfunctions

Table 1–7 (Cont.) ACP—QIO Attributes

Attribute Name ¹	Maximum Size (bytes)	Meaning
ATR\$_ASCNAME	20	Disk: file name, type, and version, in ASCII, including punctuation. Format: name.type;version. Magnetic tape: contains 17-character file identifier (ANSI a); no version number. Overrides all other file name and file type specifications if supplied on input operations. If specified on an access operation and you want only a value to be returned, specify (in ATR\$_W_SIZE) a buffer of greater than 17 bytes.
ATR\$_CREDATE ²	8	64-bit creation date and time.
ATR\$_REVDATE ^{2 3}	8	64-bit revision date and time.
ATR\$_EXPDATE ²	8	64-bit expiration date and time.
ATR\$_BAKDATE ^{3 10}	8	64-bit backup date and time.
ATR\$_UIC ²	4	4-byte file owner UIC.
ATR\$_FPRO ^{2 3}	2	File protection.
ATR\$_RPRO ¹⁰	2	2-byte record protection.
ATR\$_ACLEVEL ^{2 3 10}	1	File access level.
ATR\$_SEMASK ¹⁰	8	File security mask and limit.
ATR\$_UIC_RO ⁵	4	4-byte file owner UIC.
ATR\$_DIRSEQ ¹⁰	2	Directory update sequence count.
ATR\$_BACKLINK ¹⁰	6	File back link pointer.
ATR\$_JOURNAL ¹⁰	2	Journal control flags.
ATR\$_HDR1_ACC	1	ANSI magnetic tape header label accessibility character.
ATR\$_ADDACLNT ^{7 10 11}	255	Add one or more access control entries.
ATR\$_DELACLNT ^{7 10 11}	255	Remove an access control entry.
ATR\$_MODACLNT ^{7 10 11}	255	Modify an ACL entry.
ATR\$_FNDAACLNT ^{10 11}	255	Locate an ACL entry.

¹Attributes with an ATR\$_ prefix have two names: one with the ATR\$_ prefix for the code and one with an ATR\$_\$ prefix for the size, which is not included in the list.

²Protected (can be written to only by system or owner).

³Locked (cannot be written to while the file is locked against writers).

⁵Read only.

⁷Exclusive access required. This operation does not complete successfully if other readers or writers are allowed.

¹⁰Not supported for Files–11 On-Disk Structure Level 1 or magnetic tapes.

¹¹The status from this attribute operation is returned in FIB\$_ACL_STATUS.

ACP—QIO Interface

1.3 ACP Subfunctions

Table 1–7 (Cont.) ACP—QIO Attributes

Attribute Name¹	Maximum Size (bytes)	Meaning
ATR\$_FNDACETYP ^{10 11}	255	Find a specific type of ACE.
ATR\$_DELETEACL ^{7 10 11}	255	Delete the entire ACL.
ATR\$_READACL ^{10 11}	512	Read the entire ACL or as much as will fit in the supplied buffer. Only complete ACEs are transferred. Thus, the supplied buffer can not be completely filled.
ATR\$_ACLENGTH ^{10 11}	4	Return the length of the ACL.
ATR\$_READACE ^{10 11}	255	Read a single ACE.
ATR\$_RESERVED ^{9 10}	380	Modify reserve area.
ATR\$_HIGHWATER ¹⁰	4	High-water mark (user read-only).
ATR\$_PRIVS_USED ^{8 10}	4	Privileges used to gain access.
ATR\$_MATCHING_ACE ^{8 10}	255	ACE used to gain access (if any).
ATR\$_ACCESS_MODE	1	Access mode for following attribute descriptors.
ATR\$_FILE_SPEC ¹⁰	512	Convert FID to file specification.
ATR\$_BUFFER_OFFSET ⁴	2	Offset length for ANSI magnetic tape header label buffer.

¹Attributes with an ATR\$_ prefix have two names: one with the ATR\$_ prefix for the code and one with an ATR\$\$ prefix for the size, which is not included in the list.

⁴Not supported on write operations to MTAACP; defaults are returned on read operations.

⁷Exclusive access required. This operation does not complete successfully if other readers or writers are allowed.

⁸This attribute can only be retrieved on the initial file access or create operation.

⁹The actual length available can decrease if the file is extended in a noncontiguous manner or if an ACL is applied to the file.

¹⁰Not supported for Files–11 On-Disk Structure Level 1 or magnetic tapes.

¹¹The status from this attribute operation is returned in FIB\$_ACL_STATUS.

Table 1–8 lists the bits contained in the file characteristics longword, which is read with the ATR\$_UCHAR attribute.

ACP—QIO Interface

1.3 ACP Subfunctions

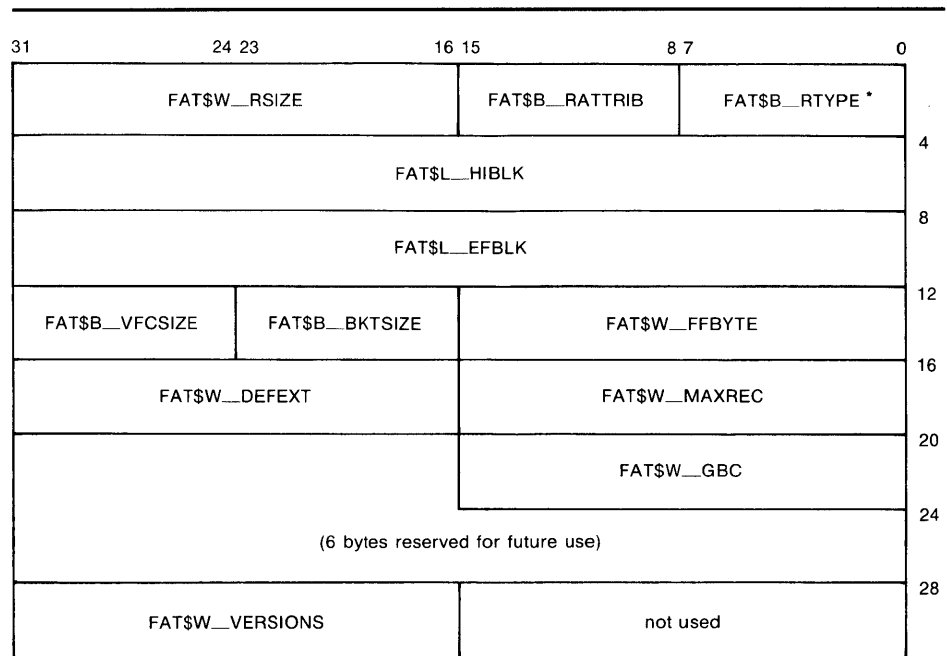
Table 1–8 File Characteristics Bits

FCHNOBACKUP	File is not to be backed up.
FCH\$_M_READCHECK	Verify all read operations.
FCH\$_M_WRITCHECK	Verify all write operations.
FCH\$_M_CONTIGB	Keep file as contiguous as possible.
FCH\$_M_LOCKED	File is deaccess-locked.
FCH\$_M_CONTIG	File is contiguous.
FCH\$_M_BADACL	File's ACL is corrupt.
FCH\$_M_SPOOL	File is an intermediate spool file.
FCH\$_M_DIRECTORY	File is a directory.
FCH\$_M_BADBLOCK	File contains bad blocks.
FCH\$_M_MARKDEL	File is marked for deletion.
FCH\$_M_ERASE	Erase file contents before deletion.

1.4 ACP QIO Record Attributes Area

Figure 1–7 shows the format of the record attributes area.

Figure 1–7 ACP—QIO Record Attributes Area



*FAT\$_V_RTYPE bits 0–3; FAT\$_V_FILEORG bits 4–7

ACP—QIO Interface

1.4 ACP QIO Record Attributes Area

Table 1–9 lists the record attributes values and their meanings.

Table 1–9 ACP Record Attributes Values

Field Value	Meaning
FAT\$B_RTYPE	Record type. Contains FAT\$_RTYPE and FAT\$_FILEORG.
FAT\$_RTYPE	Record type. The following bit values are defined: FAT\$_FIXED Fixed-length record FAT\$_VARIABLE Variable-length record FAT\$_VFC Variable-length record with fixed control FAT\$_UNDEFINED Undefined record format (stream binary) FAT\$_STREAM RMS stream format FAT\$_STREAMLF Stream terminated by LF FAT\$_STREAMCR Stream terminated by CR
FAT\$_FILEORG	File organization. The following bit values are defined: FAT\$_DIRECT Direct file organization ¹ FAT\$_INDEXED Indexed file organization FAT\$_RELATIVE Relative file organization FAT\$_SEQUENTIAL Sequential file organization
FAT\$B_RATTRIB	Record attributes. The following bit values are defined: FAT\$_FORTRANCC FORTRAN carriage control FAT\$_IMPLIEDCC Implied carriage control FAT\$_PRINTCC Print file carriage control FAT\$_NOSPAN No spanned records
FAT\$W_RSIZ	Record size in bytes.
FAT\$L_HIBLK ²	Highest allocated VBN. The ACP maintains this field when the file is extended or truncated. Attempts to modify this field in a write attributes operation are ignored. FAT\$_HIBLKH High-order 16 bits FAT\$_HIBLKL Low-order 16 bits
FAT\$L_EFBLK ^{2,3}	End-of-file VBN FAT\$_EFBLKH High-order 16 bits FAT\$_EFBLKL Low-order 16 bits
FAT\$W_FFBYTE ³	First free byte in FAT\$L_EFBLK.
FAT\$B_BKTSIZ	Bucket size in blocks.

¹Defined but not implemented.

²Inverted format field. The high- and low-order 16 bits are transposed for compatibility with PDP–11 software.

³When the end-of-file position corresponds to a block boundary, by convention FAT\$L_EFBLK contains the end-of-file VBN plus 1, and FAT\$W_FFBYTE contains 0.

ACP—QIO Interface

1.4 ACP QIO Record Attributes Area

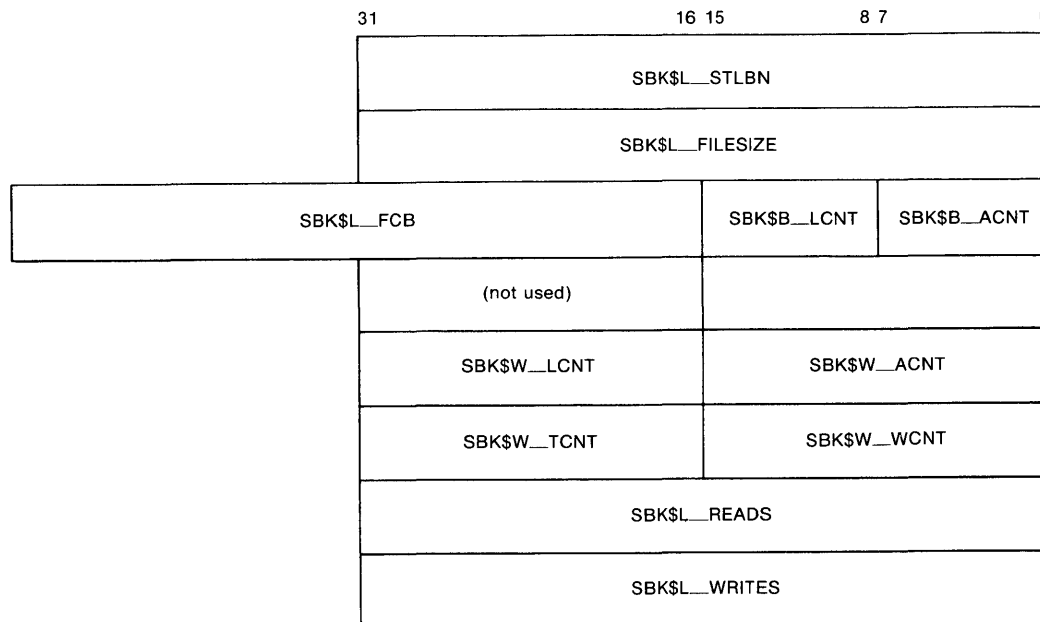
Table 1–9 (Cont.) ACP Record Attributes Values

Field Value	Meaning
FAT\$B_VFCSIZE	Size in bytes of fixed-length control for VFC records.
FAT\$W_MAXREC	Maximum record size in bytes.
FAT\$W_DEFEXT	Default extend quantity.
FAT\$W_GBC	Global buffer count.
FAT\$W_VERSIONS	Default version limit; valid only if the file is a directory.

1.5 ACP—QIO Attributes Statistics Block

Figure 1–8 shows the format of the attributes statistics block. Table 1–10 lists the contents of this block.

Figure 1–8 ACP—QIO Attributes Statistics Block



ZK-642-82

ACP—QIO Interface

1.5 ACP-QIO Attributes Statistics Block

Table 1–10 Contents of the Statistics Block

Field	Field Values	Meaning
SBK\$_STLBN		Contains the starting LBN of the file if the file is contiguous. If the file is not contiguous, this field contains a value of 0. The LBN appears as an inverted longword (the high- and low-order 16 bits are transposed for PDP-11 compatibility). The following subfields are defined:
	SBK\$_STLBNH	Starting LBN (high-order 16 bits).
	SBK\$_STLBNL	Starting LBN (low-order 16 bits).
SBK\$_FILESIZE		Contains the size of the file in blocks. The file size appears as an inverted longword (the high- and low-order 16 bits are transposed for PDP-11 compatibility). The following subfields are defined:
	SBK\$_FILESIZH	File size (high-order 16 bits).
	SBK\$_FILESIZL	File size (low-order 16 bits).
SBK\$_ACNT ¹		Access count (low byte). This field is present for PDP-11 compatibility.
SBK\$_LCNT ¹		Lock count (low byte). This field is present for PDP-11 compatibility.
SBK\$_FCB		System pool address of the file's file control block.
SBK\$_ACNT ¹		Access count (the number of channels that currently have the file open).
SBK\$_LCNT ¹		Lock count (the number of access operations that have locked the file against writers).
SBK\$_WCNT ¹		Writer count (the number of channels that currently have the file open for write).
SBK\$_TCNT ¹		Truncate lock count (the number of access operations that have locked the file against truncation).
SBK\$_READS		The number of read operations executed for the file on this channel.
SBK\$_WRITES		The number of write operations executed for the file on this channel.

¹Accesses from processes on the local node in a cluster are counted.

1.6 Major Functions

The following sections describe the operation of the major ACP functions. Each section describes the required and optional parameters for a particular function, as well as the sequence in which the function is performed. For clarity, when a major function invokes a subfunction, the input parameters used by the subfunction are omitted.

1.6.1 Create File

Create file is a virtual I/O function that creates a directory entry or a file on a disk device, or a file on a magnetic tape device.

The following is the function code:

- IO\$_CREATE

The following are the function modifiers:

- IO\$_M_CREATE—Creates a file.
- IO\$_M_ACCESS—Opens the file on your channel.
- IO\$_M_DELETE—Marks the file for deletion (applicable only to disk devices).

1.6.1.1 Input Parameters

The following are the device- or function-dependent arguments for IO\$_CREATE:

- P1—The address of the file information block (FIB) descriptor.
- P2—The address of the file name string descriptor (optional).
- 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).

The following fields in the FIB are applicable to the IO\$_CREATE operation:

Field	Field Values	Meaning
FIB\$_ACCTL		Specifies field values that control access to the file. The following bits are applicable to the IO\$_CREATE function:
	FIB\$_REWIND	Set to rewind magnetic tape before creating the file. Any data currently on the tape is overwritten.

ACP—QIO Interface

1.6 Major Functions

Field	Field Values	Meaning
	FIB\$_CURPOS	Set to create magnetic tape file at the current tape position. (Note: a magnetic tape file is created at the end of the volume set if neither FIB\$_REWIND nor FIB\$_CURPOS is set.) If the tape is not positioned at the end of a file, FIB\$_CURPOS creates a file at the next file position. Any data currently on the tape past the current file position is overwritten.
	FIB\$_WRITETHRU	Specifies that the file header is to be written back to the disk. If not specified and the file is opened, writing of the file header can be deferred to some later time.
FIB\$_CNTRLFUNC		Specifies the following value, which allows you to control actions subsequent to EOT detection on a magnetic tape file.
	FIB\$_USEREOT	Set on a per file basis to specify user EOT mode. If this bit is set, the magnetic tape driver notifies the magnetic tape system if EOT has been detected (considered a 'serious exception') during the creation of a file. The magnetic tape system, in turn, returns the alternate success code SS\$_ENDOF TAPE or SS\$_ENDOF VOLUME. All subsequent I/O requests are completed with a failure status return of SS\$_SERIOUSEXP. The driver does not execute any I/O functions until the serious exception has been explicitly cleared by issuing an IO\$_ACPCONTROL function (see Section 1.6.7). If the file is deaccessed or closed, the user EOT mode is cleared after further processing of the magnetic tape.
FIB\$_FID		Contains the file ID of the file created or entered.
FIB\$_DID		Contains the file identifier of the directory file.

ACP—QIO Interface

1.6 Major Functions

Field	Field Values	Meaning
FIB\$W_NMCTL		Controls the processing of the file name in a directory operation. The following bits are applicable to the IO\$_CREATE function:
	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_LOWVER	Set on return if a lower numbered version of the file exists. Only for disk devices.
	FIB\$M_HIGHVER	Set on return if a higher numbered version of the file exists. Only for disk devices.
FIB\$W_VERLIMIT		Specifies the version limit for the directory entry created. Used only for disk devices and only when the first version of a new file is created. If 0, the directory default is used. If a directory operation was performed, FIB\$W_VERLIMIT always contains the actual version limit of the file.
FIB\$L_ACL_STATUS		Status of the requested ACL attribute operation, if any. The ACL attributes are included in Table 1-7. If no ACL attributes are given, SS\$_NORMAL is returned here.

1.6.1.2 Disk ACP Operation

If the modifier IO\$_CREATE is specified, a file is created. The file ID of the file created is returned in FIB\$W_FID. If the modifier IO\$_DELETE is specified, the file is marked for deletion.

If a nonzero directory ID is specified in FIB\$W_DID, a directory entry is created. The file name specified by parameter P2 is entered in the directory, together with the file ID in FIB\$W_FID. (Section 1.3.1.1 describes the format for the file name string.) Wildcards are not permitted. Negative version numbers are treated as equivalent to a 0 version number. If a result string buffer and length are specified by P3 and P4, the actual file name entered, and its length, are returned.

The version number of the file receives the following treatment:

- If the version number in the specified file name is 0 or negative, the directory entry created gets a version number one greater than the highest previously existing version of that file (or version 1 if the file did not previously exist).

ACP—QIO Interface

1.6 Major Functions

- If the version number in the specified file name is a nonzero number and FIB\$M_NEWVER is set, the directory entry created gets a version number one greater than the highest previously existing version of that file, or the specified version number, whichever is greater.
- If the version number in the specified file name is a nonzero number and the directory already contains a file of the same name, type, and version, the previously existing file is set aside for deletion if FIB\$M_SUPERSEDE is specified. If FIB\$M_SUPERSEDE is not specified, the create operation fails with an SS\$_DUPFILNAM status.
- If, after creating the new directory entry, the number of versions of the file exceeds the version limit, the lowest numbered version is set aside for deletion.
- If the file did not previously exist, the new directory entry is given a version limit as follows: the version limit is taken from FIB\$W_VERLIMIT if it is a nonzero number; if it is 0, the version limit is taken from the default version limit of the directory file; if the default version limit of the directory file is 0, the version limit is set to 32,767 (the highest possible number).

The file name string entered in the directory is returned using the P3 and P4 result string parameters, if present. The file name string is also written into the header. If no directory operation was requested (FIB\$W_DID is 0), the file name string specified by P2, if any, is written into the file header.

If an attribute list is specified by P5, a write attributes subfunction is performed (see Section 1.3.5).

If the modifier IO\$_ACCESS is specified, the file is opened (see Section 1.3.2).

If the extend enable bit FIB\$M_EXTEND is specified in the FIB, an extend subfunction is performed (see Section 1.3.3).

Finally, if a file was set aside for deletion (IO\$_DELETE is specified), that file is deleted. If the file is deleted because the FIB\$M_SUPERSEDE bit was set, the alternate success status SS\$_SUPERSEDE is returned in the I/O status block. If the file is deleted because the version limit was exceeded, the alternate success status SS\$_FILEPURGED is returned.

If an error occurs in the operation of an IO\$_CREATE function, all actions performed to that point are reversed (the file is neither created nor changed), and the error status is returned to the user in the I/O status block.

1.6.1.3 Directory Entry Creation

Creating a new version of a file eliminates default access to the previously highest version of the file. For example, creating RESUME.TXT;4 masks RESUME.TXT;3 so that the DCL command TYPE RESUME.TXT yields the contents of version 4, not version 3. To protect the contents of the earlier version of a file, the creator of a file must have write access to the previous version of a file of the same name.

1.6.1.4 Magnetic Tape ACP Operation

No operation is performed unless the IO\$_CREATE modifier is specified. The magnetic tape is positioned as specified by FIB\$M_REWIND and FIB\$M_CURPOS, and the file is created. The name specified by the P2 parameter is written into the file header label.

ACP—QIO Interface

1.6 Major Functions

If P5 specifies an attribute list, a write attributes subfunction is performed (see Section 1.3.5).

If the modifier IO\$M_ACCESS is specified, the file is opened (see Section 1.3.2).

1.6.2 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 following is the function code:

- IO\$_ACCESS

The following are the function modifiers:

- IO\$M_CREATE—Creates a file.
- IO\$M_ACCESS—Opens the file on your channel.

1.6.2.1 Input Parameters

The following are the device- or function-dependent arguments for IO\$_ACCESS:

- P1—The address of the file information block (FIB) descriptor.
- P2—The address of the file name string descriptor (optional).
- 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).

ACP—QIO Interface

1.6 Major Functions

The following FIB fields are applicable to the IO\$_ACCESS operation:

Field	Field Values	Meaning
FIB\$_CNTRLFUNC		Specifies the value that allows the user to control actions subsequent to EOT detection on a magnetic tape file.
	FIB\$_USEREOT	Set on a per file basis to specify user EOT mode. If this bit is set, the magnetic tape driver notifies the magnetic tape system when EOT has been detected (considered a 'serious exception') when a file is accessed. The magnetic tape system, in turn, returns the alternate success code SS\$_ENDOFTAPE or SS\$_ENDOFVOLUME. All subsequent I/O requests are completed with a failure status return of SS\$_SERIOUSEXP. The driver does not execute any I/O functions until the serious exception has been explicitly cleared by issuing an IO\$_ACPCONTROL function (see Section 1.6.7). If the file is deaccessed or closed, the user EOT mode is cleared after further processing of the magnetic tape.
FIB\$_VERLIMIT		Receives the version limit for the file. Applicable only if FIB\$_DID is a nonzero number (if a directory lookup is done). Used only for disk devices.
FIB\$_ACL_STATUS		Status of the requested ACL attribute operation, if any. The ACL attributes are included in Table 1-7. If no ACL attributes are given, SS\$_NORMAL is returned here.
FIB\$_STATUS	FIB\$_ALT_REQ	Alternate access status. The following bits are supported: Set to indicate whether the alternate access bit is required for the current operation. If not set, the alternate access bit is optional.

ACP—QIO Interface

1.6 Major Functions

Field	Field Values	Meaning
	FIB\$_ALT_ GRANTED	If FIB\$_ALT_REQ = 0 and the alternate access check succeeded, the FIB bit returned from the file system is set.
FIB\$_ALT_ ACCESS		A 32-bit mask that represents an access mask to check against file protection; for example, to open a file for read and to check whether it can be deleted. The mask has the same configuration as the standard protection mask.

1.6.2.2 Operation

If a nonzero directory file ID is specified in FIB\$_W_DID, a lookup subfunction is performed (see Section 1.3.1.) The version limit of the file found is returned in FIB\$_W_VERLIMIT.

If the directory search fails with a 'file not found' condition and the IO\$_M_CREATE function modifier is specified, the function is reexecuted as a CREATE. In that case, the argument interpretations for IO\$_CREATE, rather than those for IO\$_ACCESS, apply.

If IO\$_M_ACCESS is specified, an access subfunction is performed to open the file (see Section 1.3.2).

If P5 specifies an attribute list, a read attributes subfunction is performed (see Section 1.3.5).

1.6.3 Deaccess File

Deaccess file is a virtual I/O function that deaccesses a file and, if specified, writes final attributes in the file header.

The following is the function code:

- IO\$_DEACCESS

IO\$_DEACCESS takes no function modifiers.

1.6.3.1 Input Parameters

The following are the device- or function-dependent arguments for IO\$_DEACCESS:

- P1—The address of the file information block (FIB) descriptor.
- P5—The address of a list of attribute descriptors (optional).

ACP—QIO Interface

1.6 Major Functions

The following FIB field is applicable to a IO\$_DEACCESS function:

Field	Field Values	Meaning
FIB\$_FID		File identification of the file being deaccessed. This field can contain a value of 0. If it does not, it must match the file identifier of the accessed file.
FIB\$_ACL_STATUS		Status of the requested ACL attribute operation, if any. The ACL attributes are included in Table 1-7. If no ACL attributes are given, SS\$_NORMAL is returned here.

1.6.3.2 Operation

For disk files, if P5 specifies an attribute control list and the file was accessed for a write operation, a write attributes subfunction is performed (see Section 1.3.5). If the file was opened for write, no attributes were specified, and FIB\$_DLOCK was set when the file was accessed, the deaccess lock bit is set in the file header, inhibiting further access to that file.

For disk files, if the truncate enable bit FIB\$_TRUNCATE is specified in the FIB, a truncate subfunction is performed (see Section 1.3.4).

Finally, the file is closed. Trailer labels are written for a magnetic tape file that was opened for write.

1.6.4 Modify File

Modify file is a virtual I/O function that modifies the file attributes or allocation of a disk file. The IO\$_MODIFY function is not applicable to magnetic tape.

The following is the function code:

- IO\$_MODIFY

IO\$_MODIFY takes no function modifiers.

1.6.4.1 Input Parameters

The following are the device- or function-dependent arguments for IO\$_MODIFY:

- P1—The address of the file information block (FIB) descriptor.
- P2—The address of the file name string descriptor (optional). If specified, the directory is searched for the name.
- 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).

ACP—QIO Interface

1.6 Major Functions

The following FIB fields are applicable to the IO\$_MODIFY function:

Field	Field Values	Meaning
FIB\$_ACCTL		Specifies field values that control access to the file. The following bits are applicable to the IO\$_MODIFY function:
	FIB\$_WRITETHRU	Specifies that the file header is to be written back to the disk. If not specified and the file is currently open, writing of the file header can be deferred to some later time.
FIB\$_VERLIMIT		If a nonzero number, specifies the version limit for the file.
FIB\$_ACL_STATUS		Status of the requested ACL attribute operation, if any. The ACL attributes are included in Table 1-7. If no ACL attributes are given, S\$_NORMAL is returned here.

1.6.4.2 Operation

If a nonzero directory ID is specified in FIB\$_DID, a lookup subfunction is executed (see Section 1.3.1). If a nonzero version limit is specified in FIB\$_VERLIMIT and the directory entry found is the latest version of that file, the version limit is set to the value specified.

If P5 specifies an attribute list, a write attributes subfunction is performed (see Section 1.3.5).

The file can be either extended or truncated. If FIB\$_EXTEND is specified in the FIB, an extend subfunction is performed (see Section 1.3.3). If FIB\$_TRUNCATE is specified in the FIB, a truncate subfunction is performed (see Section 1.3.4). Extend and truncate operations cannot be performed at the same time.

1.6.5 Delete File

Delete file is a virtual I/O function that removes a directory entry or file header from a disk volume.

The following is the function code:

- IO\$_DELETE

The following is the function modifier:

- IO\$_DELETE—Deletes the file (or marks it for deletion).

The following are the device- or function-dependent arguments for IO\$_DELETE:

- P1—The address of the file information block (FIB) descriptor.
- P2—The address of the file name string descriptor (optional).

ACP—QIO Interface

1.6 Major Functions

- 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).

The following FIB fields are applicable to the IO\$_DELETE function:

Field	Field Values	Meaning
FIB\$_ACCTL		Specifies field values that control access to the file. The following bit is applicable to the IO\$_DELETE function:
	FIB\$_WRITETHRU	Specifies that the file header is to be written back to the disk. If not specified and the file is currently open, writing of the file header can be deferred to some later time.
FIB\$_FID		Specifies the file identification to be deleted.

1.6.5.1 Operation

If a nonzero directory ID is specified in FIB\$_DID, a lookup subfunction is performed (see Section 1.3.1). The file name located is removed from the directory.

If the function modifier IO\$_DELETE is specified, the file is marked for deletion. If the file is not currently open, it is deleted immediately. If the file is open, it is deleted when the last accessor closes it.

1.6.6 Mount

Mount is a virtual I/O function that informs the ACP when a disk or magnetic tape volume is mounted. MOUNT privilege is required. IO\$_MOUNT takes no arguments or function modifiers. This function is a part of the volume mounting operation only, and it is not meant for general use. Most of the actual processing is performed by the MOUNT command or the Mount Volume (\$MOUNT) system service.

1.6.7 ACP Control

ACP Control is a virtual I/O function that performs miscellaneous control functions, depending on the arguments specified.

The following is the function code:

- IO\$_ACPCONTROL

The following is the function modifier:

- IO\$_DMOUNT—Dismounts a volume.

ACP—QIO Interface

1.6 Major Functions

1.6.7.1 Input Parameters

The following are the device- or function-dependent arguments for IO\$_ACPCONTROL:

- P1—The address of the file information block (FIB) descriptor.
- P2—The address of the file name string descriptor (optional).
- 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).

The following FIB fields control the processing of the IO\$_ACPCONTROL function:

Field	Field Values	Meaning
FIB\$_CNTRLFUNC		Specifies the control function to be performed. This field overlays FIB\$_EXCTL.
FIB\$_CNTRLVAL		Specifies additional function-dependent data. This field overlays FIB\$_EXSZ.
FIB\$_ACL_STATUS		Status of the requested ACL attribute operation, if any. The ACL attributes are included in Table 1-7. If no ACL attributes are given, SS\$_NORMAL is returned here.
FIB\$_STATUS		Alternate access status. The following bits are supported:
	FIB\$_ALT_REQ	Set to indicate whether the alternate access bit is required for the current operation. If not set, the alternate access bit is optional.
	FIB\$_ALT_GRANTED	If FIB\$_ALT_REQ = 0 and the alternate access check succeeded, the FIB bit returned from the file system is set.
FIB\$_ALT_ACCESS		A 32-bit mask that represents an access mask to check against file protection; for example, to open a file for read and to check whether it can be deleted or not. The mask has the same configuration as the standard protection mask.

ACP—QIO Interface

1.6 Major Functions

1.6.7.2 Magnetic Tape Control Functions

The following FIB field is applicable to magnetic tape operations:

Field	Field Values	Meaning
FIB\$W_CNTRLFUNC		Several ACP control functions are used for magnetic tape positioning. These functions are specified by supplying a FIB with P1 containing the FIB descriptor address. Modifiers and parameters P2, P3, and P4 are not allowed. These functions clear serious exceptions in magnetic tape drivers. The following control functions can be specified to control magnetic tape positioning:
	FIB\$_REWINDFIL	Rewind to beginning-of-file.
	FIB\$_REWINDVOL	Rewind to beginning-of-volume set.
	FIB\$_POSEND	Position to end-of-volume set.
	FIB\$_NEXTVOL	Force next volume.
	FIB\$_SPACE	Space n blocks forward or backward. The FIB\$_CNTRLVAL field specifies the number of magnetic tape blocks to space forward if positive or to space backward if negative.
	FIB\$_CLSEREXCP	If set, clears the serious exception in the magnetic tape driver (see FIB\$_USEREOT in Sections 1.6.1 and 1.6.2). This allows the user to write data blocks beyond the EOT marker, which can result in the magnetic tape not conforming to the ANSI standard for magnetic tapes (see ANSI Standard X3.27 - 1978).

1.6.7.3 Miscellaneous Disk Control Functions

Several ACP control functions are available for disk volume control. The following function does not use parameters P2, P3, and P4:

IO\$_DMOUNT	Specifying the dismount modifier on the IO\$_ACPCNTRL function executes a dismount QIO. No parameters in the FIB are used; the FIB can be omitted. This function does not perform a dismount by itself, but is used to synchronize the ACP with the DISMOUNT command and the Dismount Volume (\$DISMOUNT) system service.
-------------	---

ACP—QIO Interface

1.6 Major Functions

The FIB\$W_CNTRLFUNC field of the FIB specifies the following miscellaneous control functions (with no modifier on the IO\$_ACPCONTROL function code). These functions use no other parameters.

FIB\$_REMAP	Remap a file. The file window for the file open on the user's channel is remapped so that it maps the entire file.
FIB\$_LOCK_VOL	Allocation lock the volume. Operations that change the file structure, such as file creation, deletion, extension, and deaccess, are not permitted. If such requests are queued to the file system for an allocation-locked volume, they are not processed until the FIB\$_UNLK_VOL function is issued to unlock the volume. To issue the FIB\$_LOCK_VOL function, you must have either a system UIC or SYSPRV privilege, or be the owner of the volume.
FIB\$_UNLK_VOL	Unlock the volume. Cancels FIB\$_LOCK_VOL. To issue this function, you must have either a system UIC or SYSPRV privilege, or be the owner of the volume.

1.6.7.4 Disk Quotas

Disk quota enforcement is enabled by a quota file on the volume, or relative volume 1 if the file is on a volume set. The quota file appears in the volume's master file directory (MFD) under the name QUOTA.SYS;1. This section describes the control functions that operate on the quota file.

Table 1-11 lists the enable and disable quota control functions.

Table 1-11 Disk Quota Functions (Enable/Disable)

Value	Meaning
FIB\$_ENA_QUOTA	Enable the disk quota file. If a nonzero directory file ID is specified in FIB\$_DID, a lookup subfunction is performed to locate the quota file (see Section 1.3.1). To issue this function, you must have either a system UIC or SYSPRV privilege, or be the owner of the volume. The quota file specified by FIB\$_FID, if present, is accessed by the ACP, and quota enforcement is turned on. By convention, the quota file is named [0,0]QUOTA.SYS;1. Therefore, FIB\$_DID should contain the value 4,4,0 and the name string specified with P2 should be "QUOTA.SYS;1".
FIB\$_DSA_QUOTA	Disable the disk quota file. The quota file is deaccessed and quota enforcement is turned off. To issue this function, you must have either a system UIC or SYSPRV privilege, or be the owner of the volume.

Table 1-12 lists the quota control functions that operate on individual entries in the quota file. Each operation transfers quota file data to and from the ACP using a quota data block. This block has the same format as a record in the quota file. Figure 1-9 shows the format of this block.

ACP—QIO Interface

1.6 Major Functions

IO\$_ACPCONTROL functions that transfer quota file data between the caller and the ACP use the following device- or function-dependent arguments:

- P2—The address of a descriptor for the quota data block being sent to the ACP.
- P3—The address of a word that returns the data length.
- P4—The address of a descriptor for a buffer to receive the quota data block returned from the ACP.

Table 1–12 Disk Quota Functions (Individual Entries)

Value	Meaning
FIB\$_ADD_QUOTA	Add an entry to the disk quota file, using the UIC and quota specified in the P2 argument block. FIB\$_ADD_QUOTA requires write access to the quota file.
FIB\$_EXA_QUOTA	Examine a disk quota file entry. The entry whose UIC is specified in the P2 argument block is returned in the P4 argument block, and its length is returned in the P3 argument word. Using two flags in FIB\$_CNTRLVAL, it is possible to search through the quota file using wildcards. The two flags are: FIB\$_ALL_MEM Match all UIC members FIB\$_ALL_GRP Match all UIC groups The ACP maintains position context in FIB\$_WCC. On the first examine call, you specify 0 in FIB\$_WCC; the ACP returns a nonzero value so that each succeeding examine call returns the next matching entry. Read access to the quota file is required to examine all non-user entries.
FIB\$_MOD_QUOTA	Modify a disk quota file entry. The quota file entry specified by the UIC in the P2 argument block is modified according to the values in the block, as controlled by three flags in FIB\$_CNTRLVAL: FIB\$_MOD_ Change the permanent quota PERM FIB\$_MOD_ Change the overdraft quota OVER FIB\$_MOD_USE Change the usage data The usage data can be changed only if the volume is locked by FIB\$_LOCK_VOL (see Section 1.6.7.3). FIB\$_MOD_QUOTA requires write access to the quota file. The P3 and P4 arguments return the modified quota entry to you. By using the flags FIB\$_ALL_MEM and FIB\$_ALL_GRP, you can search through the quota file using wildcards just as you would with the FIB\$_EXA_QUOTA function.

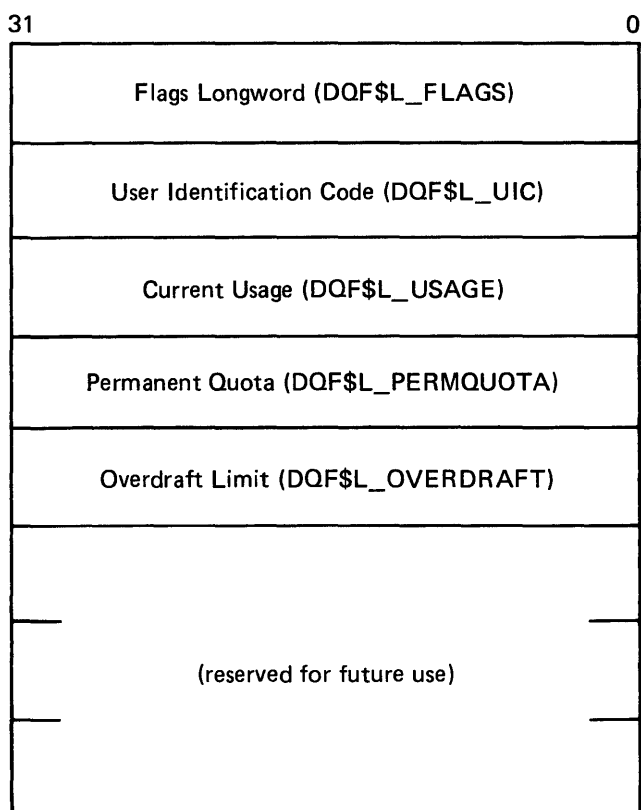
ACP—QIO Interface

1.6 Major Functions

Table 1–12 (Cont.) Disk Quota Functions (Individual Entries)

Value	Meaning
FIB\$_REM_QUOTA	<p>Remove a disk quota file entry whose UIC is specified in the P2 argument block. FIB\$_REM_QUOTA requires write access to the quota file.</p> <p>The P3 and P4 arguments return the removed quota file entry to you.</p> <p>By using the flags FIB\$_ALL_MEM and FIB\$_ALL_GRP, you can search through the quota file using wildcards just as you would with the FIB\$_EXAQUOTA function.</p>

Figure 1–9 Quota File Transfer Block



ZK-643-82

ACP—QIO Interface

1.7 I/O Status Block

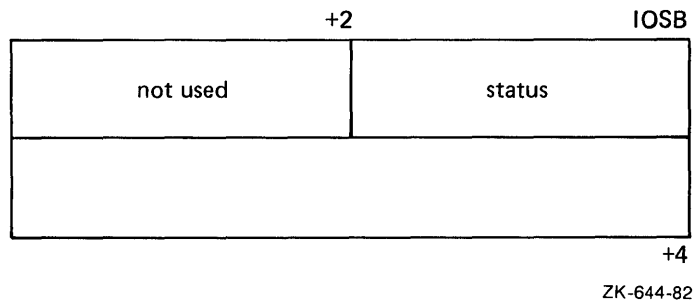
1.7 I/O Status Block

Figure 1-10 shows the I/O status block (IOSB) for ACP-QIO functions. Appendix A lists the status returns for these functions. (The *VMS System Messages and Recovery Procedures Reference Volume* provides explanations and suggested user actions for these returns.)

The file ACP returns a completion status 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\$M_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, use the values returned in FIB locations.

Figure 1-10 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.

2 Card Reader Driver

This chapter describes the use of the VMS card reader driver that supports the CR11 card reader.

2.1 Supported Card Reader Device

The CR11 card reader reads standard 80-column punched data cards.

2.2 Driver Features

The VMS card reader driver provides the following features:

- Support for 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.

The VMS operating system provides the following card reader device- or function-dependent modifier bits for read data operations:

- IO\$_M_PACKED—Read packed Hollerith code
- IO\$_M_BINARY—Read binary code

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 special card punch combination that sets the translation mode (see Section 2.2.1.2), or by the set mode function (see Section 2.4.3).

2.2.1 Special Card Punch Combinations

The 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.

Card Reader Driver

2.2 Driver Features

2.2.1.1 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 that punch combination.

2.2.1.2 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. (Table 2-5 lists the 026 and 029 punch codes.) 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.

The system can read cards that were punched on an 026 punch or an 029 punch. By default, the translation mode is 029; that is, the system reads cards from an 029 punch. However, you can change the translation mode by using the following:

- The SET CARD_READER command
- Translation mode cards

Use the SET CARD_READER command, with the /026 or /029 qualifier, to set the card reader to accept cards from either an 026 or an 029 card punch.

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.

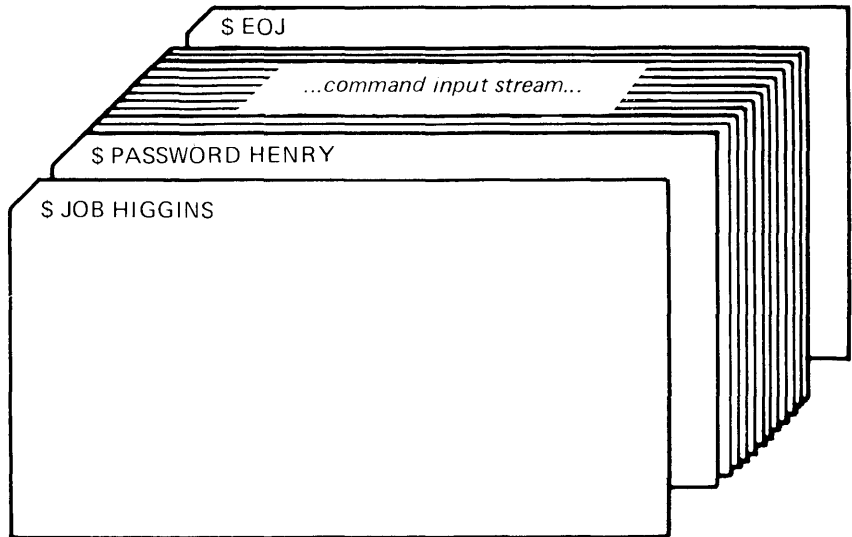
2.2.2 Submitting Batch Jobs Through the Card Reader

When you submit a batch job through a system card reader, precede the card deck containing the command procedure with cards containing JOB and PASSWORD commands. These cards specify your user name and password and, when executed, effect a login for you. The last card in the deck must contain the EOJ (End of Job) command. The EOJ card is equivalent to logging out. You can also use an overpunch card instead of an EOJ card to signal the end of a job. To do this, use an EOF card (12-11-0-1-6-7-8-9) overpunch or use the EOJ command. Figure 2-1 illustrates a card reader batch job.

Card Reader Driver

2.2 Driver Features

Figure 2-1 A Card Reader Batch Job



ZK-812-82

When the system reads a job from the card reader, it validates the user name and password specified on the JOB and PASSWORD cards. Then, it copies the entire card deck into a temporary disk file named INPBATCH.COM in your default disk and directory, and it queues the job for batch execution. Thereafter, processing is the same as for jobs submitted interactively with the SUBMIT command. When the batch job is completed, the operating system deletes the INPBATCH.COM file.

You can prevent other users from seeing your password by suppressing printing when you keypunch the PASSWORD card.

2.2.3 Passing Data to Commands and Images

To pass data to commands and images in batch jobs that you submit through a card reader, you can do the following:

- Include the data in the command procedure by placing the data on the lines after the command or image that uses the data. Use the DECK and EOD commands if the data lines begin with dollar signs.
- Temporarily redefine SYS\$INPUT as a file by using the DEFINE/USER_MODE command.

Card Reader Driver

2.2 Driver Features

2.2.4 Error Recovery

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

- If the card reader is offline 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 every 30 seconds to the system operator.
- The current operation is retried every two seconds to test for a changed situation, such as the removal of an error condition.
- The current I/O operation can be canceled at the next timeout without the card reader being online. When the card reader comes online, device operation resumes automatically.

When a recoverable card reader failure is detected and an error message is displayed on the system operator console, examine the card reader indicator lights to determine the reason for the failure. Any errors that occur must be fixed manually. The recovery is transparent to the user program issuing the I/O request.

The four categories of card reader failures and their respective recovery procedures are as follows:

- **Pick check**—The next card cannot be delivered from the input hopper to the read mechanism. To recover from this error, remove the next card to be read from the input hopper and smooth the leading edge (the edge that enters the read mechanism first). Replace the card in the input hopper and press the RESET button. The card reader operation resumes 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. To recover from this error, remove the last card read from the output hopper and examine it. If it is excessively worn or mutilated, repunch it. Place either card in the read station of the input hopper and press the RESET button. The card reader operation resumes automatically. If the stack check error recurs immediately, an adjustment is required.
- **Hopper check**—Either the input hopper is empty or the output hopper is full. To recover from this error, 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. The card reader operation resumes automatically.
- **Read check**—The last card was read incorrectly. To recover from this error, remove the last card from the output hopper and examine it. If it is excessively worn, mutilated, or contains punches before column 0 or after column 80, repunch the card. Place either card in the read station of the input hopper and press the RESET button. The card reader operation resumes automatically. If the read check error recurs immediately, an adjustment is necessary.

2.3 Device Information

You can obtain information on card reader characteristics by using the Get Device/Volume Information (\$GETDVI) system service. See the *VMS System Services Reference Manual*.

\$GETDVI returns card reader characteristics when you specify the item codes DVI\$_DEVCHAR and DVI\$_DEVDEPEND. Tables 2-1 and 2-2 list these characteristics. The \$DEVDEF macro defines the device-independent characteristics; the \$CRDEF macro defines the device-dependent characteristics.

DVI\$_DEVTYPE and DVI\$_DEVCLASS return the device type and device class names, which are defined by the \$DCDEF macro. The device class for card readers is DC\$_CARD. The device type for the CR11 is DT\$_CR11. DVI\$_DEVBUFSIZ returns the buffer size. The default buffer size to be used for all card reader devices is 80 bytes.

Table 2-1 Card Reader Device-Independent Characteristics

Characteristic ¹	Meaning
Dynamic Bit (Conditionally Set)	
DEV\$_AVL	Device is online and available
Static Bits (Always Set)	
DEV\$_IDV	Device is capable of input
DEV\$_REC	Device is record-oriented

¹Defined by the \$DEVDEF macro.

Table 2-2 Device-Dependent Characteristics for Card Readers

Value ¹	Meaning	
CR\$_TMODE	Specifies the translation mode for nonbinary, nonpacked Hollerith data transfers. ² Possible values are:	
CR\$_T026		Translate according to 026 punch code
CR\$_T029		Translate according to 029 punch code

¹Defined by the \$CRDEF macro.

²Section 2.2.1.2 describes the set translation mode punch code.

2.4 Card Reader Function Codes

The VMS card reader can perform logical, virtual, and physical I/O functions. Table 2-3 lists these functions and their function codes. These functions are described in more detail in the sections that follow.

Card Reader Driver

2.4 Card Reader Function Codes

Table 2–3 Card Reader I/O Functions

Function Code and Arguments	Type ¹	Function Modifiers	Function
IO\$_READLBLK P1,P2	L	IO\$_M_BINARY IO\$_M_PACKED	Read logical block.
IO\$_READVBLK P1,P2	V	IO\$_M_BINARY IO\$_M_PACKED	Read virtual block.
IO\$_READPBLK P1,P2	P	IO\$_M_BINARY IO\$_M_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.

¹V = virtual; L = logical; P = physical

2.4.1 Read

Read is a function that 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.

The VMS operating system provides the following read function codes:

- IO\$_READVBLK—Read virtual block
- IO\$_READLBLK—Read logical block
- IO\$_READPBLK—Read physical block

The following 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

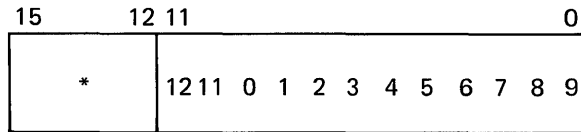
The read binary function modifier (IO\$_M_BINARY) and the read packed Hollerith function modifier (IO\$_M_PACKED) can be used with all read functions. If IO\$_M_BINARY is specified, successive columns of data are stored in sequential word locations of the input buffer. If IO\$_M_PACKED is specified,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

Card Reader Driver

2.4 Card Reader Function Codes

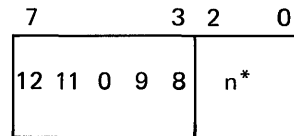
Figure 2–2 Binary and Packed Column Storage

Binary column (IO\$_M_BINARY):



*Bits 12 - 15 are 0

Packed column (IO\$_M_PACKED):



*n = 0 if no punches in rows 1 - 7
 = 1 if a punch in row 1
 = 2 if a punch in row 2
 ⋮
 = 7 if a punch in row 7

ZK-646-82

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.

2.4.2 Sense Mode

Sense mode is a function that senses the current device-dependent card reader characteristics and returns them in the second longword of the I/O status block (see Table 2–2). No device- or function-dependent arguments are used with IO\$_SENSEMODE.

2.4.3 Set Mode

Set mode operations affect the operation and characteristics of the associated card reader device. The VMS operating system defines the following types of set mode functions:

- Set mode
- Set characteristic

Card Reader Driver

2.4 Card Reader Function Codes

2.4.3.1 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. The following function code is provided.

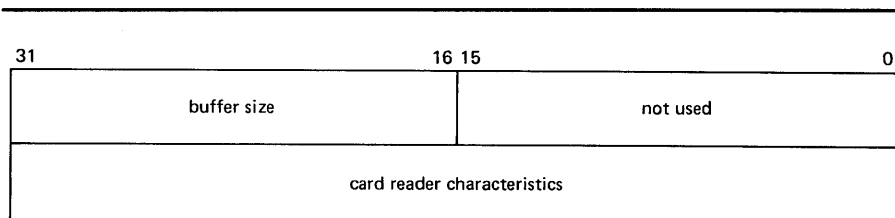
- IO\$_SETMODE

This function takes the following device- or function-dependent argument:

- P1—The address of a characteristics buffer

Figure 2-3 shows the quadword set mode characteristics buffer.

Figure 2-3 Set Mode Characteristics Buffer



ZK-647-82

Table 2-4 lists the card reader characteristics and their meanings. The \$CRDEF macro defines the characteristics values. Table 2-5 lists the 026 and 029 card reader codes.

Table 2-4 Set Mode and Set Characteristic Card Reader Characteristics

Value ¹	Meaning	
CR\$_TMODE	Specifies the translation mode for nonbinary, nonpacked Hollerith data transfers. Possible values are:	
CR\$\$_TMODE		
CR\$_T026		Translate according to 026 punch code
CR\$_T029		Translate according to 029 punch code

¹If neither the 026 nor 029 mode is specified, the default mode can be set by the SET CARD_READER command.

Table 2-5 Card Reader Codes

Character	ASCII ₈	DEC029	DEC026
{	173	12 0	12 0
}	175	11 0	11 0
SPACE	40	NONE	NONE
!	41	11 8 2	12 8 7
"	42	8 7	0 8 5

Card Reader Driver

2.4 Card Reader Function Codes

Table 2–5 (Cont.) Card Reader Codes

Character	ASCII ₈	DEC029	DEC026
—	43	8 3	0 8 6
\$	44	11 8 3	11 8 3
%	45	0 8 4	0 8 7
&	46	12	11 8 7
'	47	8 5	8 6
(50	12 8 5	0 8 4
)	51	11 8 5	12 8 4
*	52	11 8 4	11 8 4
+	53	12 8 6	12
,	54	0 8 3	0 8 3
-	55	11	11
.	56	12 8 3	12 8 3
/	57	0 1	0 1
0	60	0	0
1	61	1	1
2	62	2	2
3	63	3	3
4	64	4	4
5	65	5	5
6	66	6	6
7	67	7	7
8	70	8	8
9	71	9	9
:	72	8 2	11 8 2
;	73	11 8 6	0 8 2
<	74	12 8 4	12 8 6
=	75	8 6	8 3
>	76	0 8 6	11 8 6
?	77	0 8 7	12 8 2
@	100	8 4	8 4
A	101	12 1	12 1
B	102	12 2	12 2
C	103	12 3	12 3
D	104	12 4	12 4
E	105	12 5	12 5
F	106	12 6	12 6
G	107	12 7	12 7
H	110	12 8	12 8

Card Reader Driver

2.4 Card Reader Function Codes

Table 2–5 (Cont.) Card Reader Codes

Character	ASCII ₈	DEC029	DEC026
I	111	12 9	12 9
J	112	11 1	11 1
K	113	11 2	11 2
L	114	11 3	11 3
M	115	11 4	11 4
N	116	11 5	11 5
O	117	11 6	11 6
P	120	11 7	11 7
Q	121	11 8	11 8
R	122	11 9	11 9
S	123	0 2	0 2
T	124	0 3	0 3
U	125	0 4	0 4
V	126	0 5	0 5
W	127	0 6	0 6
X	130	0 7	0 7
Y	131	0 8	0 8
Z	132	0 9	0 9
[133	12 8 2	11 8 5
\	134	11 8 7	8 7
]	135	0 8 2	12 8 5
↑ or ^	136	12 8 7	8 5
← or _	137	0 8 5	8 2

Application programs that change specific card reader characteristics should first use the `IO$_SENSEMODE` function to read the current characteristics, modify them, and then use the set mode function to write back the results. Failure to follow this sequence results in clearing any previously set characteristic.

2.4.3.2 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. The following function code is provided:

- `IO$_SETCHAR`

This function takes the following device- or function-dependent argument:

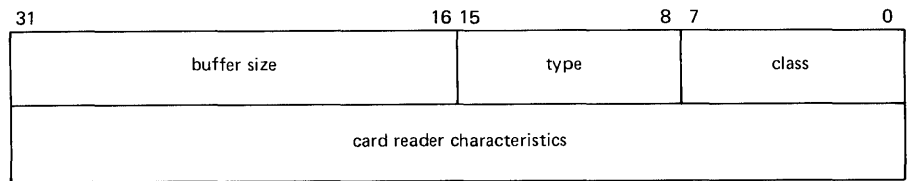
- P1—The address of a characteristics buffer

Card Reader Driver

2.4 Card Reader Function Codes

Figure 2-4 shows the set characteristic characteristics buffer.

Figure 2-4 Set Characteristic Buffer



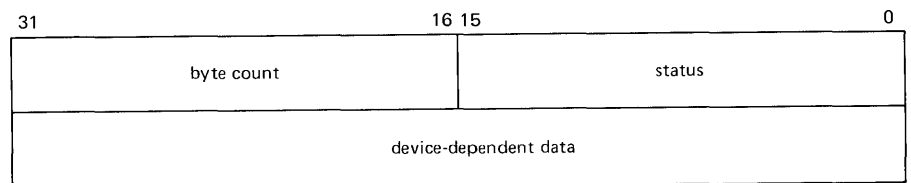
ZK-648-82

The device type value is DT\$_CR11. The device class value is DC\$_CARD. Table 2-4 lists the card reader characteristics for the Set Characteristic function.

2.5 I/O Status Block

The I/O status block (IOSB) format for QIO functions on the card reader is shown in Figure 2-5. Appendix A lists the status returns for these functions. (The *VMS System Messages and Recovery Procedures Reference volume* provides explanations and suggested user actions for these returns.) Table 2-2 lists the device-dependent data returned in the second longword. The IO\$_SENSEMODE function can be used to obtain this data.

Figure 2-5 IOSB Contents



ZK-649-82

3 Disk Drivers

This chapter describes the use of VMS disk drivers. These drivers support the devices listed in Table 3-1.

All disk drivers support Files-11 On-Disk Structure Level 1 and Level 2 file structures. Access to these file structures is through the DCL commands INITIALIZE and MOUNT, followed by the VMS RMS calls described in the *VMS Record Management Services Manual*. Files in RT-11 format can be read or written with the file exchange facility EXCHANGE.

3.1 Supported Disk Devices and Controllers

The following sections provide greater detail about the disk devices listed in Table 3-1. To obtain additional information about a device, use the DCL command SHOW DEVICE with the /FULL qualifier, the Get Device/Volume Information (\$GETDVI) system service (from a program), or the F\$GETDVI lexical function (in a command line or command procedure). Section 3.3 lists the information on disk devices returned by \$GETDVI.

Table 3-1 Supported Disk Devices

Disk Device	Code	Type	DSA	Logical Blocks/Drive
RA60	DJ	Packed	Yes	400,176
RA70	DU	Fixed	Yes	547,041
RA80	DU	Fixed	Yes	236,964
RA81	DU	Fixed	Yes	891,072
RA82	DU	Fixed	Yes	1,216,665
RB02	DQ	Cartridge	No	20,480
RB80	DQ	Fixed	No	242,606
RC25	DA	Fixed, Cartridge	Yes ¹	102,400 ²
RRD50	DU	Optical	Yes ¹	1,669,400
RD32	DU	Fixed	Yes ¹	83,204
RD51	DU	Fixed	Yes ¹	21,600
RD52	DU	Fixed	Yes ¹	60,480
RD53	DU	Fixed	Yes ¹	138,672
RD54	DU	Fixed	Yes ¹	311,200
RL02	DL	Cartridge	No	20,480
RM03	DR	Packed	No	131,680

¹Incompatible with the UDA50, KDA50 and KDB50 disk devices.

²51,200 fixed; 51,200 cartridge.

Disk Drivers

3.1 Supported Disk Devices and Controllers

Table 3–1 (Cont.) Supported Disk Devices

Disk Device	Code	Type	DSA	Logical Blocks/Drive
RM05	DR	Packed	No	500,384
RM80	DR	Fixed	No	242,606
RP05	DB	Packed	No	171,798
RP06	DB	Packed	No	340,670
RP07	DR	Fixed	No	1,008,000
RK06	DM	Cartridge	No	27,126
RK07	DM	Cartridge	No	53,790
RX01	DX	Flexible	No	494
RX02	DY	Flexible	No	494 ³ 988 ⁴
RX33	DU	Flexible	Yes ¹	2,400
RX50	DU	Flexible	Yes ¹	800
TU58 ⁵	DD	Cartridge	No	512

¹Incompatible with the UDA50, KDA50 and KDB50 disk devices.

³Single density (See Section 3.3).

⁴Double density (See Section 3.3).

⁵A magnetic tape device, the TU58 operationally resembles a disk device. See Section 3.1.16 for a description of the TU58 in disk terms.

3.1.1 UDA50 UNIBUS Disk Adapter

The UDA50 UNIBUS Disk Adapter (UDA50) is a microprocessor-based disk controller for mass storage devices that implement the DIGITAL Storage Architecture (DSA); for more information on the DSA, see Section 3.2.1.3.

The UDA50 is used to connect any combination of four RA60, RA80, and RA81 disk drives to the UNIBUS. Two UDA50 controllers can be attached to a single UNIBUS for a maximum of eight disk drives per UNIBUS. On the VAX-11/780 processor, the VMS operating system supports one UDA50 on the first UNIBUS, which can accommodate certain other options. Adding a second UDA50 requires a second UNIBUS. With the exception of the first UNIBUS, a maximum of two UDA50s per UNIBUS are supported. If two UDA50s are on a UNIBUS, no other options can be placed on that UNIBUS. The VAX-11/730 processor supports only one UDA50 per UNIBUS.

The UDA50, in implementing DSA, takes over the control of the physical disk unit. The VMS operating system processes request virtual or logical I/O on disks controlled by the UDA50. The VMS operating system maps virtual block addresses into logical block addresses. The UDA50 then resolves logical block addresses into physical block addresses on the disk.

The UDA50 corrects bad blocks on the disk by requesting that the disk class driver *revector* a failing physical block to another, error-free physical block on the disk; the logical block number is not changed (see Section 3.2.6.1). Any bad blocks might exist on a disk attached to a UDA50 are transparent to the

3.1 Supported Disk Devices and Controllers

VMS operating system, which does logical or virtual I/O to such a disk. The UDA50 also corrects most data errors.

3.1.2 KDA50 Disk Controller

The KDA50 disk controller is a two-module disk controller that allows the RA series DSA disk drives to be attached to Q-BUS systems. The KDA50 performs the same functions as the UDA50 (see Section 3.1.1).

3.1.3 KDB50 Disk Controller

The KDB50 disk controller is a two-module disk controller that allows the RA series DSA disk drives to be attached to BI bus systems, such as the VAX 8200 processor. The KDB50 performs the same functions as the UDA50 (see Section 3.1.1).

3.1.4 HSC50 Controller

The HSC50 is a high-speed, high-availability controller for mass storage devices that implement the DIGITAL Storage Architecture (DSA); for more information about the DSA, see Section 3.2.1.3. The HSC50 is connected to a processor by a Computer Interconnect (CI). The VMS operating system supports the use of the HSC50 in controlling the RA60, RA80, and RA81 disks.

The HSC50, in implementing DSA, takes over the control of the physical disk unit. VMS operating system processes request virtual or logical I/O on disks controlled by the HSC50. The VMS operating system maps virtual block addresses into logical block addresses. The HSC50 then resolves logical block addresses into physical block addresses on the disk.

The HSC50 corrects bad blocks on the disk by revectoring a failing physical block to another, error-free physical block on the disk; the logical block number is not changed. The VMS operating system, which does logical or virtual I/O to such a disk, does not recognize that any bad blocks might exist on a disk attached to an HSC50. The HSC50 also corrects most data errors.

The HSC50 provides access to disks despite most hardware failures, and it enables two or more processors to access files on the same disk.

Note: Only one system should have write access to a Files-11 On-Disk Structure Level 1 disk or to a foreign-mounted disk; all other systems should only have read access to the disk. For Files-11 On-Disk Structure Level 2 volumes, the VMS operating system enables read/write access to all nodes that are members of the same VAXcluster.

The HSC50 enables you to add or subtract disks from the device configuration without rebooting the system.

Disk Drivers

3.1 Supported Disk Devices and Controllers

3.1.5 RA60 Pack Disk

The RA60 is a large-capacity, removable disk that provides 205 MB of usable storage (7.5 million bits of data per square inch) with transfer rates of 1.9 MB/second (burst) and 950 KB/second (sustained). The RA60 belongs to the DIGITAL Storage Architecture (DSA) family of disk devices (see Section 3.2.1.3). It is connected to either a UNIBUS Disk Adapter (UDA50) or an HSC50 controller. Up to 4 disk drives can be connected to each UDA50. Up to 24 disk drives can be connected to each HSC50.

3.1.6 RB02 and RL02 Cartridge Disks

The RL02 cartridge disk is a removable, random-access mass storage device with two data surfaces. The RL02 is connected to the system by an RL11 controller that interfaces with the UNIBUS adapter. Up to four RL02 disk drives can be connected to each RL11 controller. For physical I/O transfers, the track, sector, and cylinder parameters describe a physical 256-byte RL02 sector (see Section 3.4).

When the RL02 is connected to an RB730 controller on a VAX-11/730 processor, it is identified internally as an RB02 disk drive. Disk geometry is unchanged and RL02 disk packs can be exchanged between drives on different controllers. Up to four drives can be connected to the RB730 controller.

3.1.7 RM03 and RM05 Pack Disks

The RM03 and RM05 pack disks are removable, moving-head disks that consist of five data surfaces for the RM03 and 19 data surfaces for the RM05. These disks are connected to the system by a MASSBUS adapter (MBA). Up to eight disk drives can be connected to each MBA.

3.1.8 RA80/R80/RM80 and RA81 Fixed Media Disks

The R80 is a nonremovable, large capacity, moving-head disk that consists of 14 data surfaces. Depending on how it is connected to the system, the R80 is identified internally as an RA80, RB80, or RM80, as follows:

- RA80—An R80 connected to the system through a UNIBUS disk adapter (UDA50) or an HSC50 controller. Up to four disk drives can be connected to each UDA50. Up to 24 disk drives can be connected to each HSC50.
- RB80—An R80 connected to the system through an RB730 controller on a VAX 11/730 processor. Of the maximum of four drives that can be connected to an RB730 controller, only one can be an RB80.
- RM80—An R80 connected to the system through a MASSBUS adapter (MBA). Up to eight disk drives can be connected to each MBA.

The RA81 is a large-capacity, nonremovable disk that can hold more than 890,000 blocks of data. This translates into more than 455 MB per spindle. The RA81 is connected to a UDA50 or an HSC50 controller. Up to four disk drives can be connected to each UDA50. Up to 24 drives can be connected to each HSC50.

Disk Drivers

3.1 Supported Disk Devices and Controllers

The RA80 and RA81 belong to the DIGITAL Storage Architecture (DSA) family of disk devices (see Section 3.2.1.3).

3.1.9 RP05 and RP06 Pack Disks

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

3.1.10 RP07 Fixed Media Disk

The RP07 is a 516 MB, fixed media disk drive that attaches to the MASSBUS of the VAX-11/780 system. The RP07 transfers data at 1.3 million bytes per second or as an option at a peak rate of 2.2 million bytes per second. The nine platters rotate at 3600 rpm and their data is accessed at an average speed of 31.3 milliseconds. These disks are connected to the system by an MBA. Up to eight disk drives can be connected to each MBA.

3.1.11 RK06 and RK07 Cartridge Disks

The RK06 cartridge disk is a removable, random-access, bulk storage device with three data surfaces. The RK07 cartridge disk is a double-density RK06. The RK06 and RK07 are connected to the system by an RK611 controller that interfaces to the UNIBUS adapter. Up to eight disk drives can be connected to each RK611.

3.1.12 RC25 Disk

The RC25 disk is a self-contained, Winchester-type, mass storage device that consists of a disk adapter module, a disk drive, and an integrated disk controller. The drive contains two 8-inch, double-sided disks. One of the disks (RCF25) is a sealed, nonremovable, fixed-media disk. The other disk is a removable cartridge disk that is sealed until it is loaded into the disk drive. The disks share a common drive spindle, and together they provide 52 million bytes of storage. Adapter modules interface the RC25 with either a UNIBUS system or with a Q-BUS system.

3.1.13 RRD50 Read-Only Memory (CDROM)

The RRD50 is a Compact Disc Read-Only Memory (CDROM) device that uses replicated media with a formatted capacity of 600 MB. The RRD50 consists of a table-top drive unit and a dual-board Q-BUS controller; the RRD50 subsystem is a standard disk MSCP device.

Disk Drivers

3.1 Supported Disk Devices and Controllers

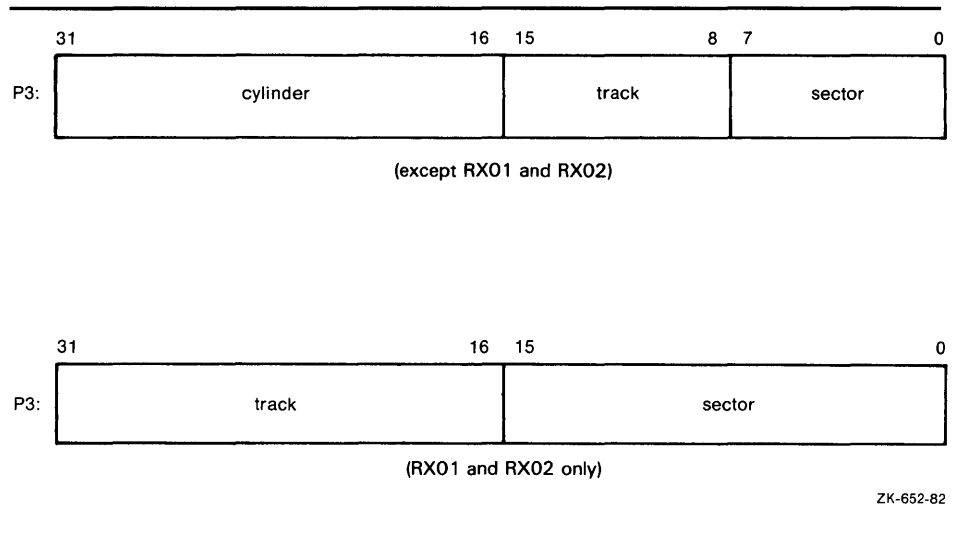
3.1.14 RX01 Console Disk

The RX01 disk uses a diskette. The disk is connected to the LSI console on the VAX-11/780, which the driver accesses using the MTPR and MFPR privileged instructions.

For logical and virtual block I/O operations, data is accessed with one block resolution (four sectors). The sector numbers are interleaved to expedite data transfers. Section 3.2.5 describes sector interleaving in greater detail.

For physical block I/O operations, the track, sector, and cylinder parameters describe a physical, 128-byte RX01 sector (see Figure 3-1 and Section 3.4). Note that the driver does not apply track-to-track skew, cylinder offset, or sector interleaving to this physical medium address.

Figure 3-1 Disk Physical Address



3.1.15 RX02 Disk

The RX02 disk is a mass storage device that uses removable diskettes. The RX02 supports existing single-density, RX01-compatible diskettes. A double-density mode allows diskettes to be recorded at twice the linear density. An entire diskette must be formatted in either single or double density. Mixed mode diskettes are not allowed.

The RX02 is connected to the system by an RX211 controller that interfaces with the UNIBUS adapter. Up to two disk drives can be controlled by each RX211.

For logical and virtual block I/O operations, data is accessed with single block resolution (four single-density sectors or two double-density sectors). The sector numbers are interleaved to expedite data transfers. Section 3.2.5 describes this feature in greater detail.

3.1 Supported Disk Devices and Controllers

For physical block I/O operations, the track and sector parameters shown in Figure 3-1 describe a physical sector (128 bytes in single density; 256 bytes in double density). The driver does not apply track-to-track skew, cylinder offset, or sector interleaving to the physical medium address.

3.1.16 TU58 Magnetic Tape (DECtape II)

The TU58 is a random-access, mass storage magnetic tape device capable of reading and writing 256K bytes per drive of data on block-addressable, preformatted cartridges at 800 bits per inch. Unlike conventional magnetic tape systems, which store information at variable positions on the tape, the TU58 stores information at fixed positions on the tape, as do magnetic disk or floppy disk devices. Thus, blocks of data can be placed on tape in a random fashion, without disturbing previously recorded data. In its physical geometry, the tape is conceptually viewed as having one cylinder, four tracks per cylinder, and 128 sectors per track. Each sector contains one 512-byte block.

The TU58 uses two vectors. NUMVEC=2 is required on the CONNECT command when specifying SYSGEN parameters.

The TU58 interfaces with the UNIBUS adapter through a DL11-series interface device. Both the TU58 and the DL11 should be set to 9600 baud. (Because the TU58 is attached to a DL11, the user cannot directly access the TU58 registers if the TU58 is on the UNIBUS.) The *DIGITAL Terminals and Communications Handbook* provides additional information on the DL11. The TU58, which has its own controller, can access either one or two tape drives.

3.2 Driver Features

VMS disk drivers provide the following features:

- Multiple controllers of the same type (except RB730), for example, more than one MBA or RK611 can be used on the system
- Multiple disk drives per controller (The exact number depends on the controller.)
- Different types of disk drives on a single controller
- Static dual porting (MBA drives only)
- Overlapped seeks (except RL02, RX01, RX02, and TU58)
- Data checks on a per-request, per-file, or per-volume basis (except RX01 and RX02)
- Full recovery from power failure for online disk drives with volumes mounted
- Extensive error recovery algorithms, such as error code correction and offset (except RB02, RL02, RX01, RX02, and TU58); for DSA disks, these algorithms are implemented in the controller
- 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
- Online diagnostic support for drive level diagnostics

Disk Drivers

3.2 Driver Features

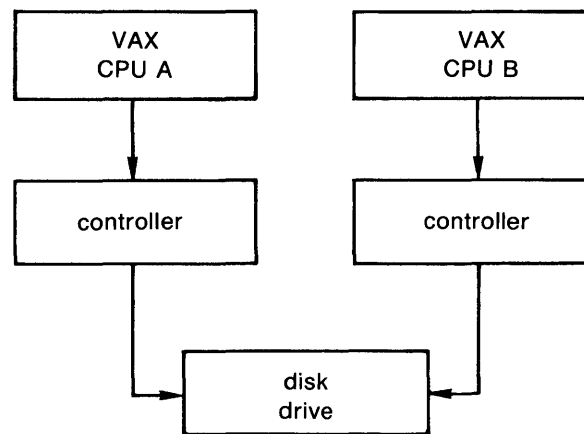
- Multiple-block, noncontiguous, virtual I/O operations at the driver level
- Logical-to-physical sector translation (RX01 and RX02 only)

The following sections describe the data check, overlapped seek, error recovery, and logical-to-physical translation features in greater detail.

3.2.1 Dual Porting (MASSBUS)

The VMS MASSBUS disk drivers, DBDRIVER and DRDRIVER, support static dual porting. Dual porting allows two MASSBUS controllers to access the same disk drive. Figure 3-2 shows this configuration. The RP05, RP06, RP07, RM03, RM05, and RM80 disk drives can be ordered, or upgraded in the field, with the MASSBUS dual port option.

Figure 3-2 Dual-Ported Disk Drives



ZK-650-82

3.2.1.1 Port Selection and Access Modes

The port select switches, on each disk drive, select the ports from which the drive can be accessed. A drive can be in one of the following access modes:

- Locked on Port A—The drive is in a single-port mode (Port A). It does not respond to any request on Port B.
- Locked on Port B—The drive is in a single-port mode (Port B). It does not respond to any request on Port A.
- Programmable A/B—The drive is capable of responding to requests on either Port A or Port B. In this mode, the drive is always in one of the following states:
 - The drive is connected and responding to a request on Port A. It is closed to requests on Port B.
 - The drive is connected and responding to a request on Port B. It is closed to requests on Port A.

Disk Drivers

3.2 Driver Features

- The drive is in a neutral state. It is equally available to requests on either port on a first-come, first-serve basis.

The operational condition of the drive cannot be changed with the port select switches after the drive becomes ready. To change from one mode to another, the drive must be in a nonrotating condition. After the new mode selection has been made, the drive must be restarted.

If a drive is in the neutral state and a disk controller either reads or writes to a drive register, the drive immediately connects a port to the requesting controller. For read operations, the drive remains connected for the duration of the operation. For write operations, the drive remains connected until a release command is issued by the device driver or a one second timeout occurs. After the connected port is released from its controller, the drive checks the other port's request flag to determine whether there has been a request on that port. If no request is pending, the drive returns to the neutral state.

3.2.1.2 Disk Use and Restrictions

If the volume is mounted foreign, read/write operations can be performed at both ports provided the user maintains control of where information is stored on the disk.

The Autoconfigure Utility currently may not be able to locate the nonactive port. For example, if a dual-ported disk drive is connected and responding at Port A, the CPU attached to Port B might not be able to find Port B with the Autoconfigure Utility. If this problem occurs, execute the AUTOCONFIGURE ALL/LOG command after the system is running.

3.2.1.2.1 Restriction on Dual-Ported Non-DSA Disks in a VAXcluster

Do not use SYSGEN to AUTOCONFIGURE or CONFIGURE a dual-ported, non-DSA disk that is already available on the system through use of an MSCP server. Establishing a local connection to the disk when a remote path is already known creates two uncoordinated paths to the same disk. Use of these two paths may corrupt files and data on any volume mounted on the drive.

Note: If the disk is not dual-ported or is never served by an MSCP server on the remote host, this restriction does not apply.

In a VAXcluster, dual-ported non-DSA disks (MASSBUS or UNIBUS) can be connected between two nodes of the cluster. These disks can also be made available to the rest of the cluster using the MSCP server on either or both of the hosts to which a disk is connected.

If the local path to the disk is not found during the bootstrap, then the MSCP server path from the other host will be the only available access to the drive. The local path will not be found during a boot if any of the following conditions exist:

- The port select switch for the drive is not enabled for this host.
- The disk, cable, or adapter hardware for the local path is broken.
- There is sufficient activity on the other port to hide the existence of the port.

Disk Drivers

3.2 Driver Features

- The system is booted in such a way that the SYSGEN AUTOCONFIGURE ALL command in the SYS\$SYSTEM:STARTUP.COM procedure was not executed.

Use of the disk is still possible through the MSCP server path.

After the configuration of the disk has reached this state, it is important **not** to add the local path back into the system I/O database. Because the VMS operating system does not provide an automatic method for adding this local path, the only possible way that you can add this local path is to use the Sysgen Utility (SYSGEN) qualifiers AUTOCONFIGURE or CONFIGURE to configure the device. SYSGEN is currently not able to detect the presence of the disk's MSCP path, and will incorrectly build a second set of data structures to describe it. Subsequent events could lead to incompatible and uncoordinated file operations, which might corrupt the volume.

To recover the local path to the disk, it is necessary to reboot the system connected to that local path.

See the *VMS VAXcluster Manual* for additional information on dual-ported disk operation.

3.2.1.3 Dual-Porting DSA Disks

Although all DSA disks (see Section 3.2.6) can be dual-ported, only one DSA controller (UDA50 or HSC50) can control a disk at a time. These disks have a DSA controller connected to each port, but control cannot be shifted from one controller to another automatically. However, if one port fails, it is possible to access the information on the disk through the other port. Except for UDASDs, the VMS operating system automatically switches access to the operational port provided the allocation class information has been set up correctly (see the *VMS VAXcluster Manual*) and the volume is mounted Files-11 (the volume is not mounted foreign).

3.2.2 Data Check

A data check is made after successful completion of a read or write operation and, except for the TU58, compares the data in memory with the data on disk to make sure they match.

Disk drivers support data checks at the following levels:

- Per request—You 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 operation. IO\$_M_DATACHECK is not supported for the RX01 and RX01 drivers.
- Per volume—You can specify the characteristics “data check all reads” and “data check all writes” when the volume is mounted. The *VMS DCL Dictionary* describes volume mounting and dismounting. The *VMS System Services Reference Manual* describes the Mount Volume (\$MOUNT) and Dismount Volume (\$DISMOU) system services.
- Per file—You can specify the file access attributes “data check on read” and “data check on write.” File access attributes are specified when the file is accessed. Chapter 1 of this manual and the *VMS Record Management Services Manual* describe file access.

Disk Drivers

3.2 Driver Features

Offset recovery is performed during a data check but Error Code Correctable (ECC) correction is not performed (see Section 3.2.4). For example, if a read operation is performed and an ECC correction is 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 the data check could not be performed because of an ECC correction.

Data checks on read operations are extremely rare, and you 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.

A data check operation directed to a TU58 does not compare the data in memory with the data on tape. Instead, either a read check or a write check operation is performed (see Sections 3.4.1 and 3.4.2).

3.2.3 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). Except on RL02, RX01, RX02, TU58 drives, MicroVAX 2000, VAXstation 2000, or on controllers with floppy disks (for example, RQDX3) when the disk is doing I/O, seek operations can be overlapped. That is, when 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.

All DSA controllers perform extensive seek optimization functions as part of their operation; IO\$M_INHSEEK has no effect on these controllers.

3.2.4 Error Recovery

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

- Handling special conditions such as power failure and interrupt timeout.
- Retrying nonfatal controller and drive errors. For DSA disks, this function is implemented by the controller.
- Applying error correction information (not applicable for RB02, RL02, RX01, RX02, and TU58). For DSA disks, error correction is implemented by the controller.
- Offsetting read heads to try to obtain a stronger recorded signal (not applicable for RB02, RL02, RB80, RM80, RX01, RX02, and TU58). For DSA disks, this function is implemented by the controller.

Disk Drivers

3.2 Driver Features

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 online, followed by reexecution 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 reexecuted up to eight times before a timeout error is returned.

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

All normal error recovery procedures (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 recovery and timeout recovery.

3.2.4.1 Skip Sectoring

Skip sectoring is a bad block treatment technique implemented on R80 disk drives (the RB80 and RM80 drives). In each track of 32 sectors, one sector is reserved for bad block replacement. Consequently, an R80 drive has available only 31 sectors per track. The Get Device/Volume Information (`$GETDVI`) system service returns this value.

You can detect bad blocks when a disk is formatted. Most formatters place these blocks in a bad block file. On an R80 drive, the first bad block encountered on a track is designated as a skip sector. This is accomplished by setting a flag in the sector header on the disk and placing the block in the skip sector file.

When a skip sector is encountered during a data transfer, it is skipped over, and all remaining blocks in the track are shifted by one physical block. For example, if block number 10 is a skip sector, and a transfer request was made beginning at block 8 for four blocks, then blocks 8, 9, 11, and 12 will be transferred. Block 10 will be "skipped."

Because skip sectors are implemented at the device driver level, they are not visible to you. The device appears to have 31 contiguous sectors per track. Sector 32 is not directly addressable, although it is accessed if a skip sector is present on the track.

3.2.5 Logical-to-Physical Translation (RX01 and RX02)

Logical block-to-physical sector translation on RX01 and RX02 drives adheres to the standard VMS format. For each 512-byte logical block selected, the driver reads or writes four 128-byte physical sectors (or two 256-byte physical sectors if an RX02 is in double-density mode). To minimize rotational latency, the physical sectors are interleaved. Interleaving allows the processor time to complete a sector transfer before the next sector in the block reaches the read/write heads. To allow for track-to-track switch time, the next logical sector that falls on a new track is skewed by six sectors. (There is no interleaving or skewing on read physical block and write physical block I/O operations.) Logical blocks are allocated starting at track 1; track 0 is not used.

Disk Drivers

3.2 Driver Features

The translation procedure, in more precise terms, is as follows:

- 1 Compute an uncorrected medium address using the following dimensions:

Number of sectors per track = 26

Number of tracks per cylinder = 1

Number of cylinders per disk = 77

- 2 Correct the computed address for interleaving and track-to-track skew (in that order) as shown in the following VAX FORTRAN statements. ISECT is the sector address and ICYL is the cylinder address computed in step 1:

Interleaving:

```
ITEMP = ISECT*2
```

```
IF (ISECT .GT. 12) ITEMP = ITEMP-25
```

```
ISECT = ITEMP
```

Skew:

```
ISECT = ISECT+(6*ICYL)
```

```
ISECT = MOD (ISECT, 26)
```

- 3 Set the sector number in the range of 1 through 26 as required by the hardware:

```
ISECT = ISECT+1
```

- 4 Adjust the cylinder number to cylinder 1 (cylinder 0 is not used):

```
ICYL = ICYL+1
```

3.2.6 DIGITAL Storage Architecture (DSA) Devices

The DIGITAL Storage Architecture (DSA) is a collection of specifications that cover all aspects of a mass storage product. The specifications are grouped into the following general categories:

- Media format—Describes the structure of sectors on a disk and the algorithms for replacing bad blocks
- Drive-to-controller interconnect—Describes the connection between an RA60, RA80, or RA81 drive and its controller
- Controller-to-host communications—Describes how hosts request controllers to transfer data

Because the VMS operating system supports all DSA disks, it supports all controller-to-host aspects of DSA. Some of these disks, such as the RA60, RA80, and RA81, use the standard drive-to-controller specifications. Other disks, such as the RC25, RD51, RD52, RD53, and RX50, do not. Disk systems that use the standard drive-to-controller specifications employ the same hardware connections and use the HSC50 and UDA50 interchangeably. Disk systems that do not use the drive-to-controller specifications provide their own internal controller, which conforms to the controller-to-host specifications.

Disk Drivers

3.2 Driver Features

DSA disks differ from MASSBUS and UNIBUS disks in the following ways:

- DSA disks contain no bad blocks. The hardware and the disk class driver (DUDRIVER) function to ensure a logically contiguous range of good blocks. If any block in the user area of the disk develops a defective area, all further access to that block is revectorred to a spare good block. Consequently, it is never necessary to run the Bad Block Locator Utility (BAD) on DSA disks. There is no manufacturer's bad block list and the file BADBLK.SYS is empty. (The Verify Utility, which is invoked by the ANALYZE /DISK_STRUCTURE /READ_CHECK command, can be used to check the integrity of newly received disks.) See Section 3.2.6.1 for additional information about bad block replacement for DSA disks.
- Insert a WAIT statement in your SYSTARTUP_V50.COM file prior to the first MOUNT statement for a DSA disk. The wait period should be about two to four seconds for the UDA50 and about 30 seconds for the HSC50. The wait time is controller-dependent and can be as much as several minutes if the controller is offline or otherwise inoperative. If this wait is omitted, the MOUNT request may fail with a "no such device" status.
- The DUDRIVER and the DSA device controllers allow multiple, concurrently outstanding QIO requests. The order in which these requests complete might not be in the order in which they were issued.
- All DSA disks can be dual-ported, but only one HSC/UDA controller can control a disk at a time (see Section 3.2.1.3). DSA disks that use an internal controller cannot be dual-ported.
- In many cases, you can attach a DSA disk to its controller on a running VMS operating system and then use it immediately without manual intervention.
- DSA disks and the DUDRIVER do not accept physical QIO data transfers or seek operations.

3.2.6.1 Bad Block Replacement and Forced Errors for DSA Disks

Disks that are built according to the DSA specifications appear to be error free. Some number of logical blocks are always capable of recording data. When a disk is formatted, every user-addressable logical block is mapped to a functioning portion of the actual disk surface, which is known as a physical block. The physical block has the true data storage capacity represented by the logical block.

Additional physical blocks are set aside to replace blocks that fail during normal disk operations. These extra physical blocks are called *replacement blocks*. Whenever a physical block to which a logical block is mapped begins to fail, the associated logical block is remapped (revectorred) to one of the replacement blocks. The process that revectorred logical blocks is called a *badblock replacement* operation. Bad block replacement operations use data stored in a special area of the disk called the *Replacement and Caching Table (RCT)*.

When a drive-dependent error threshold is reached, the need for a bad block replacement operation is declared. Depending on the controller involved, the bad block replacement operation is performed either by the controller itself (as is the case with HSCs) or by the host (as is the case with UDAs). In either case, the same steps are performed. After inspecting and altering the RCT, the failing block is read and its contents are stored in a reserved section of the RCT.

Disk Drivers

3.2 Driver Features

The design goal of DSA disks is that this read operation proceeds without error and that the RCT copy of the data is correct (as it was originally written). The failing block is then tested with one or more data patterns. If no errors are encountered in this test, the original data is copied back to the original block and no further action is taken. If the data-pattern test fails, the logical block is revector to a replacement block. After the block is revector, the original data is copied back to the revector logical block. In all these cases, the original data is preserved and the bad block replacement operation occurs without the user being aware that it happened.

However, if the original data cannot be read from the failing block, a best attempt copy of the data is stored in the RCT and the bad block replacement operation proceeds. When the time comes to write-back the original data, the best attempt data (stored in the RCT) is written back with the *forced error* flag set. The forced error flag is a signal that the data read is questionable. Reading a block that contains a forced error flag causes the status SS\$_FORCEDERROR to be returned. This status is displayed by the following message:

```
%SYSTEM-F-FORCEDERROR, forced error flagged in last sector read
```

Writing into a block always clears the forced error flag.

Note that most VMS utilities and DCL commands treat the forced error flag as a fatal error and terminate operation when they encounter it. However, the Backup Utility (BACKUP) continues to operate in the presence of most errors, including the forced error. BACKUP continues to process the file, and the forced error flag is lost. Thus, data that was formerly marked as questionable may become "correct" data.

System managers (and other users of BACKUP) should assume that forced errors reported by BACKUP signal possible degradation of the data.

To determine what, if any, blocks on a given disk volume have the forced error flag set, use the ANALYZE /DISK_STRUCTURE /READ_CHECK command, which invokes the Verify Utility. The Verify Utility reads every logical block allocated to every file on the disk and then reports (but ignores) any forced error blocks encountered.

3.2.7 VAXstation 2000 and MicroVAX 2000 Disk Driver

The VAXstation 2000 and MicroVAX 2000 disk driver supports some DSA disk operation. In particular, the driver supports block revectoring and bad block replacement. This provides the system with a logically perfect disk medium.

Like other DSA disks, if a serious error occurs during a replacement operation, the disk is write-locked to prevent further changes. This is done to preserve data integrity and minimize damage that could be caused by failing hardware. Unlike other DSA disks, there is no visible indication on the drive itself that the disk is write-locked. However, the following indicators help you determine that the disk has become write-protected:

- ERRFMT messages show that the disk is write-locked.
- The disk enters mount verification and hangs.
- DCL command SHOW DEVICE output shows that the disk is write-locked.

Disk Drivers

3.2 Driver Features

- Error messages from programs and utilities attempting to write to the disk.

If the disk becomes write-locked, you should use the following procedure:

- 1 Shut down the system.
- 2 Use standalone BACKUP to create a full backup of the disk.
- 3 Format the disk with the disk formatter.
- 4 Restore the disk from the backup using standalone BACKUP. Note that any files with sectors flagged with a forced error may be corrupted and may need to be restored from a previous backup.

If errors occurring during replacement operations persist, call DIGITAL Field Service.

3.3 Device Information

You can obtain information on all disk device characteristics by using the Get Device/Volume Information (\$GETDVI) system service (see the *VMS System Services Reference Manual*).

\$GETDVI returns disk characteristics when you specify the item codes DVI\$_DEVCHAR and DVI\$_DEVCHAR2. Table 3-2 lists the possible characteristics for disk devices.

Table 3-2 Disk Device Characteristics

Characteristic ¹	Meaning
Dynamic Bits (Conditionally Set)	
DEV\$_AVL	Device is online and available.
DEV\$_CDP ²	Dual-path device with two UCBs.
DEV\$_CLU ²	Device is available clusterwide.
DEV\$_2P ²	Device is dual-pathed.
DEV\$_FOR	Device is foreign.
DEV\$_MNT	Volume is mounted.
DEV\$_RCK	Perform data check all reads.
DEV\$_WCK	Perform data check all writes.
DEV\$_MSCP ²	Device is accessed using the mass storage control protocol.
DEV\$_RCT	Disk contains Replacement and Caching Table.
DEV\$_SRV ²	For a VAXcluster, device is served by the MSCP server.

¹Defined by the \$DEVDEF macro

²These bits are located in DVI\$_DEVCHAR2.

Disk Drivers

3.3 Device Information

Table 3–2 (Cont.) Disk Device Characteristics

Characteristic ¹	Meaning
Static Bits (Always Set)	
DEV\$_FOD	Device is file-oriented.
DEV\$_IDV	Device is capable of input.
DEV\$_ODV	Device is capable of output.
DEV\$_RND	Device is capable of random access.
DEV\$_SHR	Device is shareable.

¹Defined by the \$DEVDEF macro

DVI\$_DEVBUFSIZ returns the buffer size. The buffer size is the default to be used for disk transfers (this default is normally 512 bytes). DVI\$_DEVTYPE and DVI\$_DEVCLASS return the device type and class names, which are defined by the \$DCDEF macro. The disk model determines the device type. For example, the device type for the RA81 is DT\$_RA81. (Foreign device types take the form DT\$_FD1 through DT\$_FD8.) The device class for disks is DC\$_DISK.

DVI\$_CYLINDERS returns the number of cylinders per volume (that is, per disk), DVI\$_TRACKS returns the number of tracks per cylinder, and DVI\$_SECTORS returns the number of sectors per track. Values are returned as four-byte decimal numbers.

DVI\$_MAXBLOCK returns the maximum number of blocks (1 block = 512 bytes) that can be contained on the volume (that is, on the disk). Values are returned as four-byte decimal numbers. This information can be used, for example, to determine the density of an RX02 diskette (single density = 494 blocks, double density = 988 blocks).

3.4 Disk Function Codes

VMS disk drivers can perform logical, virtual, and physical I/O functions. Foreign-mounted devices do not require privilege to perform logical and virtual I/O requests.

Logical and physical I/O functions allow access to volume storage and require only that the issuing process have access to the volume. However, DSA disks and the disk class driver (DUDRIVER) do not accept physical QIO data transfers or seek operations.

Note: The results of logical and physical I/O operations are unpredictable if an ancillary control process (ACP) or extended QIO processing (XQP) is present.

Virtual I/O functions require an ACP for Files–11 On-Disk Structure Level 1 files or an XQP for Files–11 On-Disk Structure Level 2 files. Virtual I/O functions must be executed in a prescribed order. First, you create and access a file, then you write information to that file, and lastly you deaccess the file. Subsequently, when you access the file, you read the information, and then deaccess the file. Delete the file when the information is no longer useful.

Disk Drivers

3.4 Disk Function Codes

Non-DSA disk devices can read or write up to 65,535 bytes in a single request. DSA devices connected to an HSC50 can transfer up to 4 billion bytes in a single request. In all cases, the maximum size of the transfer is limited by the number of pages that can be faulted into the process's working set, and then locked into physical memory. (The disk driver is responsible for any memory management functions of this type.) The size of the transfer does not affect the applicable quotas (direct I/O count, buffered I/O count, and AST count limit). These quotas refer to the number of outstanding I/O operations of each type, not the size of the I/O operation being performed.

The volume to which a logical or virtual function is directed must be mounted for the function actually to 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. Chapter 1 describes the QIO level interface to the disk device ACP.

Table 3-3 Disk I/O Functions

Function Code and Arguments	Type ¹	Function Modifiers	Function
IO\$_CREATE P1,[P2],[-P3],[P4],[P5]	V	IO\$_CREATE IO\$_ACCESS IO\$_DELETE	Create a directory entry or a file.
IO\$_ACCESS P1, [P2],[-P3],[P4],[P5]	V	IO\$_CREATE IO\$_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 or allocation, or both.
IO\$_DELETE P1,[P2],[-P3],[P4],[P5]	V	IO\$_DELETE	Remove a directory entry or file header, or both.
IO\$_ACPCONTROL P1,-[P2],[P3],[P4],[P5]	V	IO\$_DMOUNT	Perform miscellaneous control functions.
IO\$_READVBLK P1,P2,P3	V	IO\$_ DATACHECK ² IO\$_INHRETRY	Read virtual block.
IO\$_READLBLK P1,P2,P3	L	IO\$_ DATACHECK ² IO\$_INHRETRY	Read logical block.
IO\$_READPBLK P1,P2,P3	P	IO\$_ DATACHECK ² IO\$_INHRETRY IO\$_INHSEEK ³	Read physical block. ⁵

¹V = virtual; L = logical; P = physical

²Not for RX01 and RX02

³Not for TU58, RX01, RX02, RB02, and RLO2

⁵Not for DSA disks

Disk Drivers

3.4 Disk Function Codes

Table 3–3 (Cont.) Disk I/O Functions

Function Code and Arguments	Type ¹	Function Modifiers	Function
IO\$_WRITEVBLK P1,P2,P3	V	IO\$_ DATACHECK ² IO\$_ERASE IO\$_INHRETRY	Write virtual block.
IO\$_WRITELBLK P1,P2,P3	L	IO\$_ DATACHECK ² IO\$_ERASE IO\$_INHRETRY	Write logical block.
IO\$_WRITEPBLK P1,P2,P3	P	IO\$_ DATACHECK ² IO\$_ERASE IO\$_INHRETRY IO\$_INHSEEK ³ IO\$_DELDATA ⁴	Write physical block. ⁵
IO\$_WRITECHECK P1,- P2,P3	P		Verify data written to disk by a previous write QIO. ²
IO\$_SENSEMODE	L		Sense the device-dependent characteristics and return them in the I/O status block.
IO\$_SENSECHAR	P		Sense the device-dependent characteristics and return them in the I/O status block.
IO\$_FORMAT P1	P		Set density (RX02 only).
IO\$_SEARCH P1	P		Search for specified block or sector (only for TU58).
IO\$_PACKACK	P		Update UCB fields if RX02; initialize volume valid on other devices. Bring DSA units online.
IO\$_AVAILABLE	P		Clear volume valid; make DSA units available.
IO\$_UNLOAD	P		Clear volume valid; make DSA units available and spin down the volume.
IO\$_SEEK P1	P		Seek to specified cylinder. ⁵

¹V = virtual; L = logical; P = physical

²Not for RX01 and RX02

³Not for TU58, RX01, RX02, RB02, and RL02

⁴RX02 only

⁵Not for DSA disks

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

- 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).

Disk Drivers

3.4 Disk Function Codes

- 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).

See Chapter 1 for more information on these functions.

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

- P1—The starting virtual address of the buffer that is to receive the data from 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, RL11, RX211, or UDA50.
- P3—The starting virtual/logical disk address of the data to be transferred in a read operation; or, in a write operation, the disk address of the area that is to receive the data.

In a virtual read or write operation, 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 using mapping windows that are set up by the file system ACP process.)

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

The function-dependent arguments for IO\$_WRITEVBLK, IO\$_WRITELBLK, and IO\$_WRITEPBLK functions that include the IO\$_M_ERASE function modifier are as follows:

- P1—The starting virtual address of the buffer that contains a four-byte, user-specified erase pattern. If the P1 address is 0, a longword of 0 will be used for the erase pattern. If the P1 address is nonzero, the contents of the four bytes starting at that address will be used as the erase pattern. DIGITAL recommends that the user specify a P1 address of 0 to lower system overhead.

Note: DSA disk controllers provide controlled, assisted erasing for the IO\$_M_ERASE modifier (with virtual and logical write functions) only when the erase pattern is all 0s. If a nonzero erase pattern is used, there is a significant performance degradation with these disks. DSA disks do not accept physical QIO transfers.

- P2—The number of bytes of erase pattern to write to the disk. The number specified is rounded up to the next highest block boundary (512 bytes).
- P3—The starting virtual, logical, or physical disk address of the data to be erased.

Disk Drivers

3.4 Disk Function Codes

The function-dependent arguments for IO\$_WRITECHECK, IO\$_READPBLK, and IO\$_WRITEPBLK are as follows:

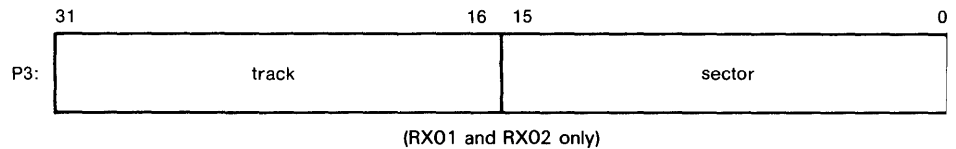
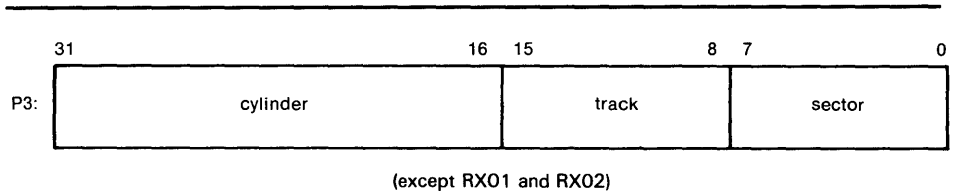
- P1—The starting virtual address of the buffer that is to receive the data in a read operation; or, in a write operation, the starting virtual address of the buffer that is to be written on the disk. Passed by reference.
- P2—The number of bytes that are to be read from the disk, or written from memory to the disk. Passed by value. An even number must be specified if the controller is an RK611, RL11, or UDA50.
- P3—The starting physical disk address of the data to be read in a read operation; or, in a write operation, the starting physical address of the disk area that is to receive the data. Passed by value. The address is expressed as sector, track, and cylinder in the format shown in Figure 3-3. (On the RX01 and RX02, the high word specifies the track number rather than the cylinder number.) Check the UCB of a currently mounted device to determine the maximum physical address value for that type of device.

Note: On the RB80 and RM80, do not address cylinders 560 and 561. These two cylinders are used for diagnostic testing only.

The function-dependent argument for IO\$_SEARCH is as follows:

- P1—The physical disk address where the tape is positioned. The address is expressed as sector, track, and cylinder in the format shown in Figure 3-3.

Figure 3-3 Starting Physical Address



ZK-652-82

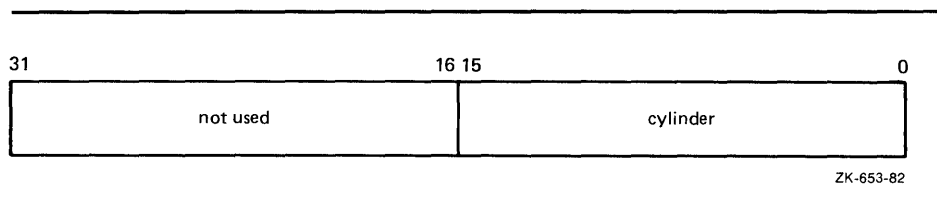
The function-dependent argument for IO\$_SEEK is as follows:

- P1—The physical cylinder number where the disk heads are positioned. The address is expressed in the format shown in Figure 3-4.

Disk Drivers

3.4 Disk Function Codes

Figure 3-4 Physical Cylinder Number Format



The function dependent argument for IO\$_FORMAT is as follows:

- P1—The density at which an RX02 diskette is reformatted (see Section 3.4.4).

3.4.1 Read

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

The VMS operating system provides the following 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 directed to a volume that is mounted foreign, that function is converted to read logical block. If a read virtual block function is directed to a volume that is mounted structured, the volume is handled in the same way as for a file-structured device.

Three function-dependent arguments are used with these codes: P1, P2, and P3. These arguments are described in 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 operation completes. A data check operation is also performed if the volume that has been 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 RX01 and RX02 drivers do not support the data check function.

If IO\$_M_DATACHECK is specified with a read function code to a TU58, or if the volume read has the characteristic "data check all reads," a read check operation is performed. This alters certain TU58 hardware parameters when the tape is read. (The read threshold in the data recovery circuit is increased; if the tape has any weak spots, errors are detected.)

The data check function modifier to a disk or tape can return five error codes in the I/O status block:

SS\$_CTRLERR SS\$_DRVERR SS\$_MEDOFL
SS\$_NONEXDRV SS\$_NORMAL

If no errors are detected, the disk or tape data is considered reliable.

Disk Drivers

3.4 Disk Function Codes

The inhibit retry function modifier (IO\$_INHRETRY) can be used with all read functions. If this modifier is specified, all error recovery attempts are inhibited. IO\$_INHRETRY takes precedence over IO\$_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.

3.4.2 Write

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

The VMS operating system provides the following 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 directed to a volume that is mounted foreign, the function is converted to write logical block. If a write virtual block function is directed to a volume that is mounted structured, the volume is handled in the same way as for a file-structured device.

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

The data check function modifier (IO\$_DATACHECK) can be used with all write operations. If this modifier is specified, a data check operation is performed after the write operation completes. 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 RX01 and RX02 drivers do not support the data check function.

If IO\$_DATACHECK is specified with a write function code to a TU58, or if the volume written has the characteristic “data check all writes,” a write check operation is performed. The write check verifies data written on the tape. First, the specified data is written on the tape. Then the tape is reversed and the TU58 controller reads the data internally to perform a checksum verification. If the checksum verification is unsuccessful after eight attempts, the write check operation is aborted and an error status is returned.

The inhibit retry function modifier (IO\$_INHRETRY) can be used with all write functions. If that modifier is specified, all error recovery attempts are inhibited. IO\$_INHRETRY takes precedence over IO\$_DATACHECK. If both IO\$_INHRETRY and IO\$_DATACHECK 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. IO\$_INHRETRY has no affect on DSA disks.

The write deleted data function modifier (IO\$_DELDATA) can be used with the write physical block (IO\$_WRITEPBLK) function to the RX02. If this modifier is specified, a deleted data address mark instead of the standard data address mark is written preceding the data. Otherwise, the operation of the IO\$_WRITEPBLK function is the same; write data is transferred to the disk. When a successful read operation is performed on this data, the status code

Disk Drivers

3.4 Disk Function Codes

SS\$_RDDELDATA is returned in the I/O status block rather than the usual SS\$_NORMAL status code.

The IO\$_M_ERASE function modifier can be used with all write function codes to erase a user-selected part of a disk. This modifier propagates an erase pattern through the specified range. Section 3.4 describes the write function arguments to be used with IO\$_M_ERASE.

3.4.3 Sense Mode

Sense mode operations obtain current disk device-dependent characteristics that are returned to the caller in the second longword of the I/O status block (see Figure 3-6). The VMS operating system provides the following function codes:

- IO\$_SENSEMODE—Sense characteristics
- IO\$_SENSECHAR—Sense characteristics

IO\$_SENSEMODE is a logical function. IO\$_SENSECHAR is a physical I/O function and requires the access privilege necessary to perform physical I/O. No device- or function-dependent arguments are used with either function.

3.4.4 Set Density

The set density function assigns a new density to an entire RX02 floppy diskette. The diskette is also reformatted: new data address marks are written (single or double density) and all data fields are zeroed. Set density is a physical I/O function and requires the access privilege necessary to perform physical I/O. The following function code is provided:

- IO\$_FORMAT

IO\$_FORMAT takes the following function-dependent argument:

- P1—The density at which the diskette is reformatted:
 - 0 = single density (default)
 - 1 = single density
 - 2 = double density

The set density operation should not be interrupted before it is completed (about 15 seconds). If the operation is interrupted, the resulting diskette might contain illegal data address marks in both densities. The diskette must then be completely reformatted and the function reissued.

3.4.5 Search

The search function positions a TU58 magnetic tape to the block specified. Search is a physical I/O function and requires the access privilege necessary to perform physical I/O. The VMS operating system provides a single function code:

- IO\$_SEARCH

This function code takes the following function-dependent argument:

- P1—Specifies the block where the read/write head will be positioned. The low byte contains the sector number in the range 0 to 127; the high byte contains the track number in the range 0 to 3.

IO\$_SEARCH can save time between read and write operations. For example, nearly 30 seconds are required to completely rewind a tape. If the last read or write operation is near the end of the tape and the next operation is near the beginning of the tape, the search operation can begin after the last operation completes, and the tape will rewind while the process is otherwise occupied. (The search QIO is not completed until the search is completed. Consequently, if a \$QIOW system service request is issued, the process will be held up until the search is completed.)

3.4.6 Pack Acknowledge

The pack acknowledge function sets the volume valid bit for all disk devices. Pack acknowledge is a physical I/O function and requires the access privilege to perform physical I/O. If directed to an RX02 drive, pack acknowledge also determines the diskette density and updates the device-dependent information returned by \$GETDVI item codes DVI\$_CYLINDERS, DVI\$_TRACKS, DVI\$_SECTORS, DVI\$_DEVTYPE, DVI\$_CLASS, and DVI\$_MAXBLOCK. If directed to a DSA disk, pack acknowledge also sends the online packet to the controller. The following function code is provided:

- IO\$_PACKACK

This function code takes no function-dependent arguments.

IO\$_PACKACK must be the first function issued when a volume (pack, cartridge, or diskette) is placed in a disk drive. IO\$_PACKACK is issued automatically when the DCL commands INITIALIZE or MOUNT are issued.

For DSA disks, the IO\$_PACKACK function involves at least one disk read and four disk writes. It also locks the drive's port selector on the port that initiated the pack acknowledge function.

3.4.7 Unload

The unload function clears the volume valid bit for all disk drives, makes DSA disks available, and issues an unload command to the drive (spins down the volume). The unload function reverses the function performed by pack acknowledge (see Section 3.4.6). The following function code is provided:

- IO\$_UNLOAD

This function takes no function-dependent arguments.

Disk Drivers

3.4 Disk Function Codes

3.4.8 Available

The available function clears the volume valid bit for all disk drives; that is, it reverses the function performed by pack acknowledge (see Section 3.4.6). No unload function is issued to the drive. Therefore, those drives capable of spinning down do not spin down. The following function code is provided:

- IO\$_AVAILABLE

This function takes no function-dependent arguments.

3.4.9 Seek

The seek function directs the read/write heads to move to the cylinder specified in the P1 argument (see Sections 3.2.3 and 3.4, and Figure 3-4).

3.4.10 Write Check

The write check function verifies that data was written to disk correctly. The data to be checked is addressed using physical disk addressing (sector, track, and cylinder) (see Figure 3-3). If the request is directed to a DSA disk, you must specify a logical block number, even though IO\$_WRITECHECK is a physical I/O function. The following function code is provided:

- IO\$_WRITECHECK

A write QIO must be used to write data to disk before you enter this command. IO\$_WRITECHECK then reads the same block of data and compares it with the data in the specified buffer. Three function-dependent arguments are used with this code: P1, P2, and P3. These arguments are described in Section 3.4.

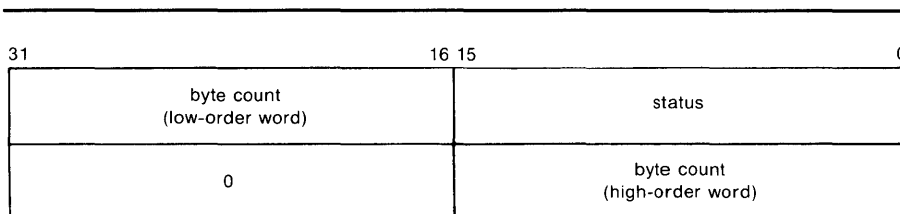
IO\$_WRITECHECK is similar to the IO\$_M_DATACHECK function modifier for write QIOs, except that IO\$_WRITECHECK does not write the data to disk; it is specified after data is written by a separate write QIO. Nonprivileged processes can use the IO\$_M_DATACHECK modifier with IO\$_WRITEVBLK (which does not require access privilege) to determine whether data is written correctly. The RX01 and RX02 drivers do not support the write check function.

The write check function and the data check function modifier to a TU58 can return six error codes in the I/O status block: SS\$_NORMAL, SS\$_CTRLERR, SS\$_DRVERR, SS\$_MEDOFL, SS\$_NONEXDRV, and SS\$_WRTLCK.

Disk Drivers

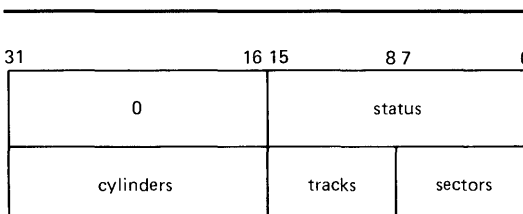
3.5 I/O Status Block

Figure 3–5 IOSB Contents



ZK-656-82

Figure 3–6 IOSB Contents - Sense Mode



ZK-657-82

3.5 I/O Status Block

Figure 3–5 shows the I/O status block (IOSB) for all disk device QIO functions except Sense Mode. Figure 3–6 shows the I/O status block for the Sense Mode function. Appendix A lists the status messages for all functions and devices. (The *VMS System Messages and Recovery Procedures Reference Volume* provides explanations and suggested user actions for these messages.)

The byte count is a 32-bit integer giving the actual number of bytes transferred to or from the process buffer.

The second longword of the I/O status block for the Sense Mode function returns information on the cylinder, track, and sector configuration for the particular device.

3.6 Programming Example

This sample program (Example 3–1) provides an example of optimizing access time to a disk file. The program creates a file using VMS 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 (\$QIO) system service.

Disk Drivers

3.6 Programming Example

Example 3-1 Disk Program Example

```

; *****
;
; .TITLE Disk Driver Programming Example
; .IDENT /01/
;
; Define necessary symbols.
;
; $FIBDEF ;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:
;
$FAB ALQ = 100,- ;Initial file size is to be
- ;100 blocks
FAC = PUT,- ;File Access Type is output
FNA = FILE_NAME,- ;File name string address
FNS = FILE_SIZE,- ;File name string size
FOP = CTG,- ;File is to be contiguous
MRS = 512,- ;Maximum record size is 512
- ;bytes
NAM = NAM_BLOCK,- ;File name block address
ORG = SEQ,- ;File organization is to be
- ;sequential
REM = FIX ;Record format is fixed length
;
; Allocate file information block.
;
; A file information block is required as an argument in the
; Queue I/O system service call that accesses a file.
;
FIB_BLOCK:
;
.BLKB FIB$K_LENGTH ;
;
; Allocate file information block descriptor.
;
FIB_DESCR:
;
.LONG FIB$K_LENGTH ;Length of the file
;information block
.LONG FIB_BLOCK ;Address of the file
;information block

```

Example 3-1 Cont'd. on next page

Disk Drivers

3.6 Programming Example

Example 3-1 (Cont.) Disk Program Example

```
;
; Allocate File Name Block
;
;   A file name block is required by RMS-32 to return information
;   concerning a file (for example, the resultant file name string
;   after logical name translation and defaults have been applied).
;
NAM_BLOCK:
    $NAM
;
; Allocate Record Access Block
;
;   A record access block is required by RMS-32 for record
;   operations on a file.
;
RAB_BLOCK:
    $RAB    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 address buffer
;
BLOCK_BUFFER:
    .BLKB  1024                   ;Direct access buffer is 1024
                                   ;bytes
;
; Allocate space to store channel number returned by the $ASSIGN
; Channel system service.
;
DEVICE_CHANNEL:
    .BLKW  1
;
; Allocate device name string and descriptor.
;
DEVICE_DESCR:
    .LONG  20$-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.
;
```

Example 3-1 Cont'd. on next page

Disk Drivers

3.6 Programming Example

Example 3-1 (Cont.) Disk Program Example

```
IO_STATUS:                                ;
    .BLKQ  1                               ;
;
; Allocate output record buffer.
;
RECORD_BUFFER:                             ;
    .BLKB  512                             ;Record buffer is 512 bytes
;
; *****
;
;                               Start Program
;
; *****
;
; The purpose of the program is to create a file called MYDATAFIL.DAT
; using RMS-32; store information concerning the file; write 100
; records, each containing its record number in every byte;
; close the file; and then access, 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:                                     ;First part of example
    $CREATE FAB = FAB_BLOCK                ;Create and open file
    BLBC    R0,20$                          ;If low bit = 0, creation
;                                               ;failure
;
; Second, connect the record access block to the created file.
;
    $CONNECT RAB = RAB_BLOCK                ;Connect the record access
;                                               ;block
    BLBC    R0,30$                          ;If low bit = 0, creation
;                                               ;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
        MOVC5  #0,(R6),R5,#512,RECORD_BUFFER ;Fill record buffer
;
; Now use RMS-32 to write the record into the newly created file.
;
```

Example 3-1 Cont'd. on next page

Disk Drivers

3.6 Programming Example

Example 3-1 (Cont.) Disk Program Example

```

$PUT   RAB = RAB_BLOCK           ;Put record in file
BLBC   RO,30$                    ;If low bit = 0, 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+NAM$W_FID,FIB_BLOCK+FIB$W_FID ;Save file
;identification
MOVW   NAM_BLOCK+NAM$W_FID+2,FIB_BLOCK+FIB$W_FID+2 ;Save
;sequence number
MOVW   NAM_BLOCK+NAM$W_FID+4,FIB_BLOCK+FIB$W_FID+4 ;Save
;relative volume
$CLOSE FAB = FAB_BLOCK           ;Close file
BLBS   RO,PART_2                 ;If low bit set, successful
;close
20$    RET                       ;Return with RMS error status
;
; Record stream connection or put record failure.
;
; Close file and return status.
;
30$:   PUSHL   RO                 ;Save error status
$CLOSE FAB = FAB_BLOCK           ;Close file
POPL   RO                       ;Retrieve error status
RET    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:
$ASSIGN_S DEVNAM = DEVICE_DESCR,- ;Assign a channel to file
CHAN = DEVICE_CHANNEL             ;device
BLBC   RO,20$                    ;If low bit = 0, assign
;failure
MOVL   #FIB$M_NOWRITE!FIB$M_WRITE,- ;Set for read/write
FIB_BLOCK+FIB$L_ACCTL             ;access
$QIOW_S CHAN = DEVICE_CHANNEL,- ;Access file on device channel
FUNC = #IO$_ACCESS!IO$M_ACCESS,- ;I/O function is
- ;access file
IO_SB = IO_STATUS,-              ;Address of I/O status
- ;quadword
P1 = FIB_DESCR                   ;Address of information block
;descriptor
BLBC   RO,10$                    ;If low bit = 0, access
;failure
MOVZWL IO_STATUS,RO              ;Get final I/O completion
;status

```

Example 3-1 Cont'd. on next page

Disk Drivers

3.6 Programming Example

Example 3-1 (Cont.) Disk Program Example

```
        BLBS    RO,30$                ;If low bit set, successful
                                           ;I/O function
10$:    PUSHL   RO                    ;Save error status
        $DASSGN_S CHAN = DEVICE_CHANNEL ;Deassign file device channel
        POPL    RO                    ;Retrieve error status
20$:    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 two 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 having the old contents of record 1, and so forth. After
; the example has been run, it is suggested that the file dump
; utility be used to verify the change in data positioning.
;
30$:    MOVZBL  #1,R6                  ;Set starting record (block)
                                           ;number
;
; Read next two records into block buffer.
;
40$:    $QIO_S CHAN = DEVICE_CHANNEL,- ;Read next two records from
        -                                           ;file channel
        FUNC = #IO$_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.
;
```

Example 3-1 Cont'd. on next page

Disk Drivers

3.6 Programming Example

Example 3-1 (Cont.) Disk Program Example

```

$QIOW_S CHAN = DEVICE_CHANNEL,- ;Write even-numbered record in
- ;odd slot
FUNC = #IO$_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
P2 = #512,- ;Length of even record buffer
P3 = R6 ;Record number of odd record
BSBB 50$ ;Check I/O completion status
ADDL3 #1,R6,R5 ;Calculate even record number
$QIOW_S CHAN = DEVICE_CHANNEL,- ;Write odd numbered record in
- ;even slot
FUNC = #IO$_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
BSBB 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 = 0, service
;failure
MOVZWL IO_STATUS,R0 ;Get final I/O completion
;status
BLBC R0,70$ ;If low bit = 0, I/O function
RSB ;failure
;
; 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
$QIOW_S CHAN = DEVICE_CHANNEL,- ;Deaccess file
FUNC = #IO$_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

```

4 Laboratory Peripheral Accelerator Driver

This chapter describes the VMS laboratory peripheral accelerator (LPA11-K) driver and the high-level language procedure library that interfaces with it. The procedure library is implemented with callable assembly language routines that translate arguments into the format required by the LPA11-K driver and that handle buffer chaining operations. Routines for loading the microcode and initializing the device are also described.

Refer to the *LPA11-K Laboratory Peripheral Accelerator User's Guide* for additional information.

4.1 Supported Device

The LPA11-K is a peripheral device that controls analog-to-digital (A/D) and digital-to-analog (D/A) converters, digital I/O registers, and real-time clocks. It is connected to the VAX processor through the UNIBUS adapter.

The LPA11-K is a fast, flexible microprocessor subsystem designed for applications requiring high-speed, concurrent data acquisition and data reduction. The LPA11-K allows aggregate analog input and output rates of up to 150,000 samples per second. The maximum aggregate digital input and output rate is 15,000 samples per second.

Table 4-1 lists the useful minimum and maximum LPA11-K configurations supported by the VMS operating system.

4.1.1 LPA11-K Modes of Operation

The LPA11-K operates in two modes: dedicated and multirequest.

In dedicated mode, only one user (one request), can be active at a time, and only analog I/O data transfers are supported. Up to two A/D converters can be controlled simultaneously. One D/A converter can be controlled at a time. Sampling is initiated either by an overflow of the real-time clock or by an externally supplied signal. Dedicated mode provides sampling rates of up to 150,000 samples per second.

Table 4-1 Minimum and Maximum Configurations per LPA11-K

Minimum	Maximum
1 DD11-Cx or Dx backplane	2 DD11-Cx or Dx backplanes
1 KW11-K real-time clock	1 KW11-K real-time clock
1 of the following:	2 AD11-K A/D converters
AD11-K A/D converter	2 AM11-K multiplexers for AD11-K converters
AA11-K A/D converter	1 AA11-K D/A converter
DR11-K digital I/O register	5 DR11-K digital I/O registers

Laboratory Peripheral Accelerator Driver

4.1 Supported Device

In multirequest mode, sampling from all of the devices listed in Table 4-1 is supported. The LPA11-K operates like a multicontroller device; up to eight requests (from one through eight users) can be active simultaneously. The sampling rate for each user is a multiple of the common real-time clock rate. Independent rates can be maintained for each user. Both the sampling rate and the device type are specified as part of each data transfer request. Multirequest mode provides a maximum aggregate sampling rate of 15,000 samples per second.

4.1.2 Errors

The LPA11-K returns the following classes of errors:

- 1 Errors associated with the issuance of a new LPA11-K command (SS\$_DEVCMDEERR)
- 2 Errors associated with an active data transfer request (SS\$_DEVREQERR)
- 3 Fatal hardware errors that affect all LPA11-K activity (SS\$_CTRLERR)

Appendix A of the *LPA11-K Laboratory Peripheral Accelerator User's Guide* lists these three classes of errors and the specific error codes for each class. The LPA11-K aborts all active requests if any of the following conditions occur:

- Power failure
- Device timeout
- Fatal error

Power failure is reported to any active users when power is recovered.

The LADRIVER times out all \$QIOs after two seconds if they have not completed. The driver does not provide any parameters that allow the user to change the length of the timeout.

The timeout period applied to all \$QIOs can be changed with the following PATCH commands executed from a privileged account:

```
$ PATCH SYS$SYSTEM:LADRIVER.EXE/OUTPUT=SYS$SYSTEM:LADRIVER.EXE
PATCH>SET ECO 25
PATCH>REPLACE/INSTRUCTION LA$TIMEOUT_VALUE
OLD>'PUSHL      I`#00000002'
OLD>EXIT
NEW>'PUSHL      I`#0000003C'
NEW>EXIT
PATCH>UPDATE
PATCH>EXIT
```

Substitute the desired timeout value for the "0000003C" in the example above. When you reboot, the system loads the new copy of the driver containing the new timeout value.

Device timeouts are monitored only when a new command is issued. For data transfers, the time between buffer full interrupts is not defined. Thus, no timeout errors are reported on a buffer-to-buffer basis.

If a required resource is not available to a process, an error message is returned immediately. The driver does not place the process in the resource wait mode.

Laboratory Peripheral Accelerator Driver

4.2 Supporting Software

4.2 Supporting Software

The LPA11-K is supported by a device driver, a high-level language procedure library of support routines, and routines for loading the microcode and initializing the device. The system software and support routines provide a control path for synchronizing the use of buffers, specifying requests, and starting and stopping requests; the actual data algorithms for the laboratory data acquisition I/O devices are accomplished by the LPA11-K.

The LPA11-K driver and the associated I/O interface have the following features:

- They permit multiple LPA11-K subsystems on a single UNIBUS adapter.
- They operate as an integral part of the VMS operating system.
- They can be loaded on a running VMS operating system without relinking the executive.
- They handle I/O requests, function dispatching, UNIBUS adapter map allocation, interrupts, and error reporting for multiple LPA11-K subsystems.
- The LPA11-K functions as a multibuffered device. Up to eight buffer areas can be defined per request. Up to eight requests can be handled simultaneously. Buffer areas can be reused after the data they contain is processed.
- Because the LPA11-K chains buffer areas automatically, a start data transfer request can transfer an infinite and noninterrupted amount of data.
- Multiple ASTs are dynamically queued by the driver to indicate when a buffer has been filled (the data is available for processing) or emptied (the buffer is available for new data).

The high-level language support routines have the following features:

- They translate arguments provided in the high-level language calls into the format required for the Queue I/O interface.
- They provide a buffer chaining capability for a multibuffering environment by maintaining queues of used, in use, and available buffers.
- They adhere to all VMS conventions for calling sequences, use of shareable resources, and reentrancy.
- They can be part of a resident global library, or they can be linked into a process image as needed.

The routines for loading microcode and initializing devices have the following features:

- They execute, as separate processes, images that issue I/O requests. These I/O requests initiate microcode image loading, start the LPA11-K subsystem, and automatically configure the peripheral devices on the LPA11-K internal I/O bus.

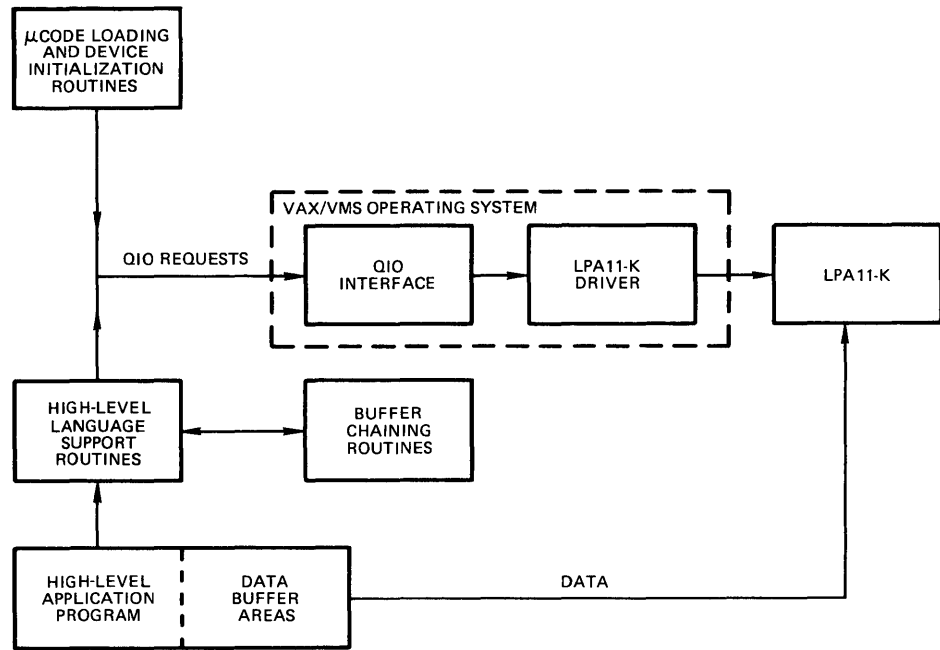
Laboratory Peripheral Accelerator Driver

4.2 Supporting Software

- They can be executed at the request of the user or an operator.
- They can be executed at the request of other processes.
- They can be executed automatically when the system is initialized and on power recovery.

Figure 4–1 shows the relationship of the supporting software to the LPA11-K.

Figure 4–1 Relationship of Supporting Software to LPA11-K



ZK-658-82

4.3 Device Information

You can obtain information on all peripheral data acquisition devices on the LPA11-K internal I/O bus by using the Get Volume Information (\$GETDVI) system service. (See the *VMS System Services Reference Manual*.)

\$GETDVI returns device characteristics when you specify the item codes DVI\$_DEVCHAR and DVI\$_DEVDEPEND. Tables 4–2 and 4–3 list these characteristics. The \$DEVDEF macro defines the device-independent characteristics; the \$LADEF macro defines the device-dependent characteristics. Device-dependent characteristics are set by the set clock, initialize, and load microcode I/O functions to any one of, or a combination of, the values listed in Table 4–3.

DVI\$_DEVCLASS and DVI\$_DEVTYPE return the device class and device type names, which are defined by the \$DCDEF macro. The device class for the LPA11-K is DC\$_REALTIME; the device type is DT\$_LPA11. DVI\$_DEVBUFSIZ is not applicable to the LPA11-K.

Laboratory Peripheral Accelerator Driver

4.3 Device Information

Table 4–2 LPA11-K Device-Independent Characteristics

Characteristic ¹	Meaning
Dynamic Bit (Conditionally Set)	
DEV\$M_AVL	Device is online and available.
Static Bits (Always Set)	
DEV\$M_IDV	Device is capable of input.
DEV\$M_ODV	Device is capable of output.
DEV\$M_RTM	Device is real-time.
DEV\$M_SHR	Device is shareable.

¹Defined by the \$DEVDEF macro.

Table 4–3 LPA11-K Device-Dependent Characteristics

Field ¹	Meaning								
LA\$M_MCVVALID LA\$V_MCVVALID	The load microcode I/O function (IO\$_LOADMCODE) was performed successfully. LA\$M_MCVVALID is set by IO\$_LOADMCODE. Each microword is verified by reading it back and comparing it with the specified value. LA\$M_MCVVALID is cleared if there is no match.								
LA\$V_MCTYPE LA\$\$_MCTYPE	The microcode type, set by the load microcode I/O function (IO\$_LOADMCODE), is one of the following values:								
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>LA\$K_MRMCODE</td> <td>Microcode type is in multirequest mode.</td> </tr> <tr> <td>LA\$K_ADMCODE</td> <td>Microcode type is in dedicated A/D mode.</td> </tr> <tr> <td>LA\$K_DAMCODE</td> <td>Microcode type is in dedicated D/A mode.</td> </tr> </tbody> </table>	Value	Meaning	LA\$K_MRMCODE	Microcode type is in multirequest mode.	LA\$K_ADMCODE	Microcode type is in dedicated A/D mode.	LA\$K_DAMCODE	Microcode type is in dedicated D/A mode.
Value	Meaning								
LA\$K_MRMCODE	Microcode type is in multirequest mode.								
LA\$K_ADMCODE	Microcode type is in dedicated A/D mode.								
LA\$K_DAMCODE	Microcode type is in dedicated D/A mode.								

¹Defined by the \$LADEF macro.

Laboratory Peripheral Accelerator Driver

4.3 Device Information

Table 4–3 (Cont.) LPA11-K Device-Dependent Characteristics

Field ¹	Meaning																						
LA\$_CONFIG LA\$\$_CONFIG	The bit positions, set by the initialize I/O function (IO\$_INITIALIZE), for the peripheral data acquisition devices on the LPA11-K internal I/O bus are one or more of the following:																						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>LA\$_CLOCKA LA\$_M_CLOCKA</td> <td>Clock A</td> </tr> <tr> <td>LA\$_CLOCKB LA\$_M_CLOCKB</td> <td>Clock B</td> </tr> <tr> <td>LA\$_AD1 LA\$_M_AD1</td> <td>A/D device 1</td> </tr> <tr> <td>LA\$_AD2 LA\$_M_AD2</td> <td>A/D device 2</td> </tr> <tr> <td>LA\$_DA LA\$_M_DA</td> <td>D/A device 1</td> </tr> <tr> <td>LA\$_DIO1 LA\$_M_DIO1</td> <td>Digital I/O buffer 1</td> </tr> <tr> <td>LA\$_DIO2 LA\$_M_DIO2</td> <td>Digital I/O buffer 2</td> </tr> <tr> <td>LA\$_DIO3 LA\$_M_DIO3</td> <td>Digital I/O buffer 3</td> </tr> <tr> <td>LA\$_DIO4 LA\$_M_DIO4</td> <td>Digital I/O buffer 4</td> </tr> <tr> <td>LA\$_DIO5 LA\$_M_DIO5</td> <td>Digital I/O buffer 5</td> </tr> </tbody> </table>	Value	Meaning	LA\$_CLOCKA LA\$_M_CLOCKA	Clock A	LA\$_CLOCKB LA\$_M_CLOCKB	Clock B	LA\$_AD1 LA\$_M_AD1	A/D device 1	LA\$_AD2 LA\$_M_AD2	A/D device 2	LA\$_DA LA\$_M_DA	D/A device 1	LA\$_DIO1 LA\$_M_DIO1	Digital I/O buffer 1	LA\$_DIO2 LA\$_M_DIO2	Digital I/O buffer 2	LA\$_DIO3 LA\$_M_DIO3	Digital I/O buffer 3	LA\$_DIO4 LA\$_M_DIO4	Digital I/O buffer 4	LA\$_DIO5 LA\$_M_DIO5	Digital I/O buffer 5
Value	Meaning																						
LA\$_CLOCKA LA\$_M_CLOCKA	Clock A																						
LA\$_CLOCKB LA\$_M_CLOCKB	Clock B																						
LA\$_AD1 LA\$_M_AD1	A/D device 1																						
LA\$_AD2 LA\$_M_AD2	A/D device 2																						
LA\$_DA LA\$_M_DA	D/A device 1																						
LA\$_DIO1 LA\$_M_DIO1	Digital I/O buffer 1																						
LA\$_DIO2 LA\$_M_DIO2	Digital I/O buffer 2																						
LA\$_DIO3 LA\$_M_DIO3	Digital I/O buffer 3																						
LA\$_DIO4 LA\$_M_DIO4	Digital I/O buffer 4																						
LA\$_DIO5 LA\$_M_DIO5	Digital I/O buffer 5																						
LA\$_RATE LA\$\$_RATE	The Clock A rate, which is set by the set clock function (IO\$_SETCLOCK), is one of the following values:																						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Stopped</td> </tr> <tr> <td>1</td> <td>1 MHz</td> </tr> <tr> <td>2</td> <td>100 kHz</td> </tr> <tr> <td>3</td> <td>10 kHz</td> </tr> <tr> <td>4</td> <td>1 kHz</td> </tr> <tr> <td>5</td> <td>100 Hz</td> </tr> <tr> <td>6</td> <td>Schmidt trigger</td> </tr> <tr> <td>7</td> <td>Line frequency</td> </tr> </tbody> </table>	Value	Meaning	0	Stopped	1	1 MHz	2	100 kHz	3	10 kHz	4	1 kHz	5	100 Hz	6	Schmidt trigger	7	Line frequency				
Value	Meaning																						
0	Stopped																						
1	1 MHz																						
2	100 kHz																						
3	10 kHz																						
4	1 kHz																						
5	100 Hz																						
6	Schmidt trigger																						
7	Line frequency																						

¹Defined by the \$LADEF macro.

Laboratory Peripheral Accelerator Driver

4.3 Device Information

Table 4–3 (Cont.) LPA11-K Device-Dependent Characteristics

Field ¹	Meaning
LA\$V_PRESET LA\$\$_PRESET	The Clock A preset value set by the set clock

¹Defined by the \$LADEF macro.

4.4 LPA11-K Function Codes

The LPA11-K I/O functions are as follows:

- Load the microcode into the LPA11-K.
- Start the LPA11-K microprocessor.
- Initialize the LPA11-K subsystem.
- Set the LPA11-K real-time clock rate.
- Start a data transfer request.

The first three functions are normally performed by the loader process, not by the user's data transfer program. See Section 4.5.21 for a description of the loader process interface.

The Cancel I/O on Channel (\$CANCEL) system service is used to abort data transfers.

4.4.1 Load Microcode

This I/O function resets the LPA11-K and loads an image of LPA11-K microcode. Physical I/O privilege is required. The following function code is provided:

- IO\$_LOADMCODE—Load microcode

The load microcode function takes the following device- or function-dependent arguments:

- P1—The starting virtual address of the microcode image that is to be loaded into the LPA11-K
- P2—The number of bytes (usually 2048) that are to be loaded
- P3—The starting microprogram address (usually 0) in the LPA11-K that is to receive the microcode

If any data transfer requests are active at the time a load microcode request is issued, the load request is rejected and SS\$_DEVACTIVE is returned in the I/O status block.

Each microword is verified by comparing it with the specified value in memory. If all words match (the microcode was loaded successfully) the driver sets the microcode valid bit (LA\$V_MCVVALID) in the device-dependent characteristics longword (see Table 4–3). If there is no match,

Laboratory Peripheral Accelerator Driver

4.4 LPA11-K Function Codes

SS\$_DATACHECK is returned in the I/O status block and LA\$_MCVALID is cleared to indicate that the microcode was not properly loaded. If the microcode was loaded successfully, the driver stores one of the microcode type values (LA\$_MRCODE, LA\$_ADCODE, or LA\$_DAMCODE) in the characteristics longword.

After a load microcode function is completed, the second word of the I/O status block contains the number of bytes loaded.

4.4.2 Start Microprocessor

This I/O function resets the LPA11-K and starts (or restarts) the LPA11-K microprocessor. Physical I/O privilege is required. The following function code is provided:

- IO\$_STARTMPROC—Start microprocessor

This function code takes no device- or function-dependent arguments.

The start microprocessor function can return five error codes in the I/O status block (see Section 4.6):

SS\$_CTRLERR	SS\$_DEVACTIVE	SS\$_MCNOTVALID
SS\$_POWERFAIL	SS\$_TIMEOUT	

Appendix A of the the *LPA11-K Laboratory Peripheral Accelerator User's Guide* provides additional information on error codes.

4.4.3 Initialize LPA11-K

This I/O function issues a subsystem initialize command to the LPA11-K. This command specifies LPA11-K laboratory I/O device addresses and other table information for the subsystem. It is issued only once after restarting the subsystem and before any other LPA11-K command is given. Physical I/O privilege is required. The VMS operating system defines the following function code:

- IO\$_INITIALIZE—Initialize LPA11-K

The initialize LPA11-K function takes the following device- or function-dependent arguments:

- P1—The starting, word-aligned, virtual address of the initialize command table in the user process. This table is read once by the LPA11-K during the execution of the initialize command. See the *LPA11-K Laboratory Peripheral Accelerator User's Guide* for additional information.
- P2—Length of the initialize command buffer (always 278 bytes).

If the initialize function is completed successfully, the appropriate device configuration values are set in the device-dependent characteristics longword (see Table 4-3).

Laboratory Peripheral Accelerator Driver

4.4 LPA11-K Function Codes

The initialize function can return the following 10 error codes in the I/O status block:

```
SS$_BUFNOTALIGN  SS$_CANCEL      SS$_CTRLERR
SS$_DEVCMDERR   SS$_INCLENGTH   SS$_INSMAPREG
SS$_IVMODE      SS$_MCNOTVALID  SS$_POWERFAIL
SS$_TIMEOUT
```

If a device specified in the initialize command table is not in the LPA11-K configuration, an error condition (SS\$_DEVCMDERR) occurs and the address of the first device not found is returned in the LPA11-K maintenance status register (see Section 4.6). A program can use this characteristic to poll the LPA11-K and determine the current device configuration.

4.4.4 Set Clock

This virtual function issues a clock control command to the LPA11-K. The clock control command specifies information necessary to start, stop, or change the sample rate at which the real-time clock runs on the LPA11-K subsystem.

Note: If the LPA11-K has more than one user, caution should be exercised when the clock rate is changed. In multirequest mode, a change in the clock rate affects all users.

The following function code is provided:

- IO\$_SETCLOCK—Set clock

The set clock function takes the following device- or function-dependent arguments:

- P2—Mode of operation. The VMS operating system defines the following clock start mode word (hexadecimal) values:

Value	Meaning
1	KW11-K Clock A
11	KW11-K Clock B

- P3—Clock control and status. The VMS operating system defines the following clock status word (hexadecimal) values:

Value	Meaning
0	Stop clock
143	1 MHz clock rate
145	100 kHz clock rate
147	10 kHz clock rate
149	1 kHz clock rate
14B	100 Hz clock rate
14D	Clock rate is Schmidt trigger 1
14F	Clock rate is line frequency

Laboratory Peripheral Accelerator Driver

4.4 LPA11-K Function Codes

- P4—The two's complement of the real-time clock preset value. The range is 16 bits for the KW11-K Clock A and 8 bits for the KW11-K Clock B.

The *LPA11-K Laboratory Peripheral Accelerator User's Guide* describes the clock start mode word and the clock status word in greater detail.

If the set clock function is completed successfully for Clock A, the clock rate and preset values are stored in the device-dependent characteristics longword (see Table 4-3).

The set clock function can return six error codes in the I/O status block (see Section 4.6):

SS\$_CANCEL	SS\$_CTRLERR	SS\$_DEVCMDDERR
SS\$_MCNOTVALID	SS\$_POWERFAIL	SS\$_TIMEOUT

Appendix A of the the *LPA11-K Laboratory Peripheral Accelerator User's Guide* provides additional information on error codes.

4.4.5 Start Data Transfer Request

This virtual I/O function issues a data transfer start command that specifies the buffer addresses, sample mode, and sample parameters used by the LPA11-K. This information is passed to the data transfer command table. The following function code is provided:

- IO\$_STARTDATA—Start data transfer request

The start data transfer request function takes the following function modifier:

- IO\$_M_SETEVF—Set event flag

The start data transfer request function takes the following device- or function-dependent arguments:

- P1—The starting virtual address of the data transfer command table in the user's process.
- P2—The length in bytes (always 40) of the data transfer command table.
- P3—The AST address of the normal buffer completion AST routine (optional).
- P4—The AST address of the buffer overrun completion AST routine (optional). This argument is used only when the buffer overrun bit (LASM_BFROVRN) is set, that is, when a buffer overrun condition is classified as a nonfatal error.

A buffer overrun condition differs from a data overrun condition. The LPA11-K fetches data from, or stores data in, memory. If data cannot be fetched quickly enough (for example, when there is too much UNIBUS activity) a data underrun condition occurs. If data cannot be stored quickly enough, a data overrun condition occurs. After each buffer is filled or emptied, the LPA11-K obtains the index number of the next buffer to process from the user status word (USW). (See Section 2.5 of the *LPA11-K Laboratory Peripheral Accelerator User's Guide*.) A buffer overrun condition occurs if the LPA11-K fills or empties buffers faster than the application program can supply new buffers. For example, buffer overrun can occur when the

Laboratory Peripheral Accelerator Driver

4.4 LPA11-K Function Codes

sampling rate is too high, the buffers are too small, or the system load is too heavy.

The LPA11-K driver accesses the 10-longword data transfer command table, shown in Figure 4–2, when the data transfer start command is processed. After the command is accepted and data transfers begin, the driver does not access the table.

Figure 4–2 Data Transfer Command Table

31	24 23	16 15	8 7	0
highest available buffer and buffer overrun bit		mode		
user status word address				
overall data buffer length				
overall data buffer address				
random channel list length				
random channel list address				
channel increment	start channel number	delay		
dwell		number of channels		
digital trigger mask		event mark channel	digital trigger channel	
		event mark mask		

ZK-660-82

In the first longword of the data transfer command table, the first two bytes contain the LPA11-K start data transfer request mode word. (Section 3.4.1 of the *LPA11-K Laboratory Peripheral Accelerator User's Guide* describes the functions of this word.)

The third byte contains the number (0–7) of the highest buffer available and the buffer overrun flag bit (bit 23; values: LA\$M_BFROVRN and LA\$V_BFROVRN). If this bit is set, a buffer overrun condition is a nonfatal error.

The second longword contains the user status word address (see Section 3.4.3 of the *LPA11-K Laboratory Peripheral Accelerator User's Guide*). This virtual address points to a two-byte area in the user-process space and must be word aligned.

Laboratory Peripheral Accelerator Driver

4.4 LPA11-K Function Codes

The third longword contains the size (in bytes) of the overall buffer area. The virtual address in the fourth longword is the beginning address of this area. This address must be longword aligned. The overall buffer area contains a specified number of buffers (the number of the highest available buffer specified in the first longword plus one). Individual buffers are subject to length restrictions: in multirequest mode the length must be in multiples of two bytes; in dedicated mode the length must be in multiples of four bytes. All data buffers are virtually contiguous for each data transfer request.

The fifth and sixth longwords contain the random channel list (RCL) length and address, respectively. The RCL address must be word aligned. The last word in the RCL must have bit 15 set. (See Section 3.4.6 of the *LPA11-K Laboratory Peripheral Accelerator User's Guide* for additional information on the RCL.)

The seventh through tenth longwords contain LPA11-K-specific sample parameters. The driver passes these parameters directly to the LPA11-K. (See the *LPA11-K Laboratory Peripheral Accelerator User's Guide* for a detailed description of their functions.)

The start data transfer request function can return the following 15 error codes in the I/O status block (see Section 4.6):

SS\$_ABORT	SS\$_BUFNOTALIGN	SS\$_CANCEL
SS\$_CTRLERR	SS\$_DEVCMDEERR	SS\$_DEVREQERR
SS\$_EXQUOTA	SS\$_INCLENGTH	SS\$_INSFBUFDP
SS\$_INSMAPREG	SS\$_INSMEM	SS\$_MCNOTVALID
SS\$_PARITY	SS\$_POWERFAIL	SS\$_TIMEOUT

Data buffers are chained and reused as the LPA11-K and the user process dispose of the data. As each buffer is filled or emptied, the LPA11-K driver notifies the application process either by setting the event flag specified by the QIO request *efn* argument or by queuing an AST. Since buffer use is a continuing process, the event flag is set or the AST is queued a number of times. The user process must clear the event flag (or receive the AST), process the data, and specify the next buffer for the LPA11-K to use.

If the set event flag function modifier (IO\$_M_SETEVF) is specified, the event flag is set repeatedly: when the data transfer request is started, after each buffer completion, and when the request completes. If IO\$_M_SETEVF is not specified, the event flag is set only when the request completes.

ASTs are preferred over event flags for synchronizing a program with the LPA11-K, because AST delivery is a queued process, while the setting of event flags is not. If only event flags are used, it is possible to lose buffer status.

Three AST addresses can be specified. For normal data buffer transactions the AST address specified in the P3 argument is used. If the buffer overrun bit in the data transfer command table is set and an overrun condition occurs, the AST address specified in the P4 argument is used. The AST address specified in the *astadr* argument of the QIO request is used when the entire data transfer request is completed. The *astprm* argument specified in the QIO request is passed to all three AST routines.

Laboratory Peripheral Accelerator Driver

4.4 LPA11-K Function Codes

If insufficient dynamic memory is available to allocate an AST block, an error (SS\$_INSMEM) is returned. If the user does not have sufficient AST quota remaining to allocate an AST block, an error (SS\$_EXQUOTA) is returned. In either case, the request is stopped. Normally, there are never more than three outstanding ASTs per LPA11-K request.

4.4.6 LPA11-K Data Transfer Stop Command

The Cancel I/O on Channel (\$CANCEL) system service is used to abort data transfers for a particular process. When the LPA11-K driver receives a \$CANCEL request, a data transfer stop command is issued to the LPA11-K.

To stop a data transfer, set bit 14 of the user status word. If this bit is set, the transfer stops at the end of the next buffer transaction (see Section 2.5 of the *LPA11-K Laboratory Peripheral Accelerator User's Guide*).

4.5 High-Level Language Interface

The VMS operating system supports several program-callable procedures that provide access to the LPA11-K. The formats of these calls are documented in this manual for VAX FORTRAN users. VAX MACRO users must set up a standard VMS argument block and issue the standard CALL procedure. (VAX MACRO users can also access the LPA11-K directly through the use of the device-specific QIO functions described in Section 4.4.) Users of other high-level languages must specify the proper subroutine or procedure invocation.

4.5.1 High-Level Language Support Routines

The VMS operating system provides 20 high-level language procedures for the LPA11-K. These procedures are divided into four classes. Table 4-4 lists the classes and the VAX procedures for the LPA11-K.

Table 4-4 VAX Procedures for the LPA11-K

Class	Subroutine	Function
Sweep Control	LPA\$ADSWP	Start A/D converter sweep
	LPA\$DASWP	Start D/A converter sweep
	LPA\$DISWP	Start digital input sweep
	LPA\$DOSWP	Start digital output sweep
	LPA\$LAMSKS	Specify LPA11-K controller and digital mask words
	LPA\$SETADC	Specify channel select parameters
	LPA\$SETIBF	Specify buffer parameters
	LPA\$STPSWP	Stop sweep
Clock control	LPA\$CLOCKA	Set Clock A rate
	LPA\$CLOCKB	Set Clock B rate
	LPA\$XRATE	Compute clock rate and preset value

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

Table 4–4 (Cont.) VAX Procedures for the LPA11-K

Class	Subroutine	Function
Data Buffer	LPA\$IBFSTS	Return buffer status
Control	LPA\$IGTBUF	Return next available buffer
	LPA\$INXTBF	Alter buffer order
	LPA\$IWTBUF	Return next buffer or wait
	LPA\$RLSBUF	Release buffer to LPA11-K
	LPA\$RMVBUF	Remove buffer from device queue
Miscellaneous	LPA\$CVADF	Convert A/D input to floating point
	LPA\$FLT16	Convert unsigned integer to floating point
	LPA\$LOADMC	Load microcode and initialize LPA11-K

4.5.1.1 Buffer Queue Control

This section is provided for informational purposes only.

Buffer queue control for data transfers by LPA11-K subroutines involves the use of the following queues:

- Device queue (DVQ)
- User queue (USQ)
- In-use queue (IUQ)

Each data transfer request can specify from one through eight data buffer areas. The user specifies these buffers by address. During execution of the request, the LPA11-K assigns an index from 0 through 7 when a buffer is referenced.

The DVQ contains the indexes of all the buffers that the user has released (buffers made available to be filled or emptied by the LPA11-K). For output functions (D/A and digital output), these buffers contain data to be output by the LPA11-K. For input functions (A/D and digital input), these buffers are empty and waiting to be filled by the LPA11-K.

The USQ contains the indexes of all buffers that are waiting to be returned to the user. The LPA\$IWTBUF and LPA\$IGTBUF calls are used to return the index of the next buffer in the USQ. For output functions (D/A and digital output), these buffers are empty and waiting to be filled by the application program. For input functions (A/D and digital input), these buffers contain data to be processed by the application program.

The IUQ contains the indexes of all buffers that are currently being processed by the LPA11-K. Normally, the IUQ contains the indexes of the following buffers:

- The buffer currently being filled or emptied by the LPA11-K
- The next buffer to be filled or emptied by the LPA11-K. (This is the buffer specified by the next buffer index field in the user status word.)

Because the LPA11-K driver requires that at least one buffer be ready when the input or output sweep is started, the user must call the LPA\$RLSBUF subroutine before the sweep is initiated.

Figure 4–3 shows the flow between the buffer queues.

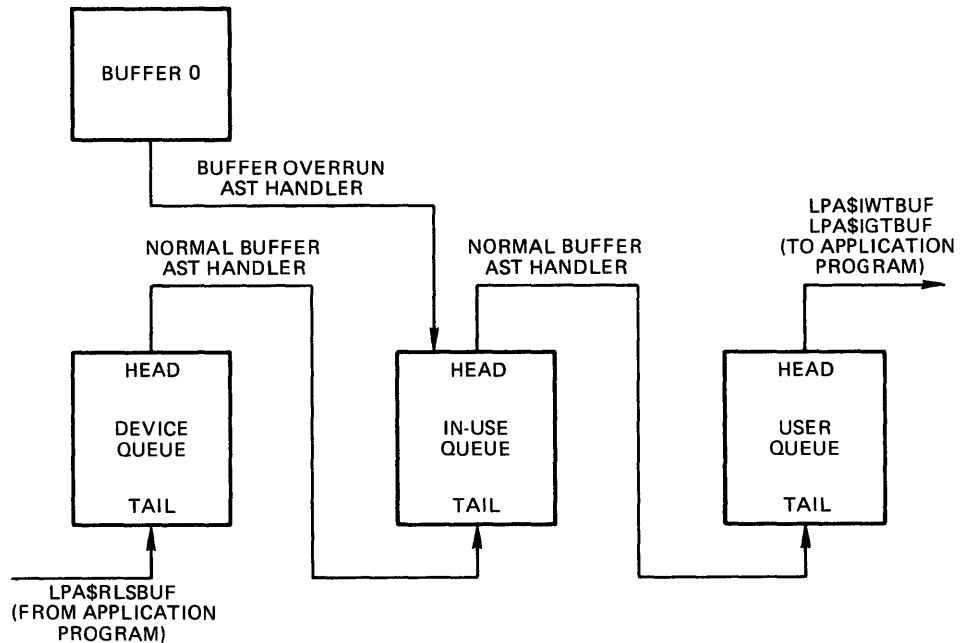
Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

4.5.1.2 Subroutine Argument Usage

Table 4-5 describes the general use of the subroutine arguments. The subroutine descriptions in the following sections contain additional information on argument usage. The (IBUF), (BUF), and (ICHN) (random channel list address) arguments must be aligned on specific boundaries.

Figure 4-3 Buffer Queue Control



ZK-661-8

Table 4-5 Subroutine Argument Usage

Argument	Meaning
IBUF	A 50-longword array initialized by the LPA\$SETIBF subroutine. IBUF is the impure area used by the buffer management subroutines. A unique IBUF array is required for each simultaneously active request. IBUF must be longword aligned. The first quadword in the IBUF array is an I/O status block (IOSB) for high-level language subroutines. The LPA\$IGTBUF and LPA\$IWTBUF subroutines fill this quadword with the current and completion status (see Section 4.6).
LBUF	Specifies the size of each data buffer in words (must be even for dedicated mode sweeps). All buffers are the same size. The minimum value for LBUF is 6 for multirequest mode data transfers and 258 for dedicated mode data transfers. The aggregate size of the assigned buffers must be less than 32,768 words. Thus, the maximum size of each buffer (in words) is limited to 32,768 divided by the number of buffers. The LBUF argument length is one word.

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

Table 4–5 (Cont.) Subroutine Argument Usage

Argument	Meaning	
NBUF	Specifies the number of times the buffers are to be filled during the life of the request. If 0 (default) is specified, sampling is indefinite and must be stopped with the LPA\$STPSWP subroutine. The NBUF argument length is one longword.	
MODE	Specifies sampling options. MODE bit values are listed in the appropriate subroutine descriptions. The default is 0. MODE values can be added to specify several options. No options are mutually exclusive, although not all bits can be applicable at the same time. The MODE argument length is one word.	
IRATE	Specifies the clock rate as follows:	
	Value	Meaning
	-1	Direct-coupled Schmidt trigger 1 (Clock A only)
	0	Clock B overflow or no rate
	1	1 MHz
	2	100 kHz
	3	10 kHz
	4	1 kHz
	5	100 Hz
	6	Schmidt trigger
7	Line frequency	
	The IRATE argument length is one longword.	
IPRSET	Specifies the hardware clock preset value. This value is the two's complement of the desired number of clock ticks between clock interrupts. (The maximum value is 0, the two's complement of 65,536.) IPRSET can be computed by the LPA\$XRATE subroutine. The IPRSET argument length is one word.	
DWELL	Specifies the number of hardware clock overflows between sample sequences in multirequest mode. For example, if DWELL is 20 and NCHN is 3, then after 20 clock overflows one channel is sampled on each of the next three successive overflows; no sampling occurs for the next 20 clock overflows. This allows different users to use different sample rates with the same hardware clock overflow rate. In dedicated mode, the hardware clock overflow rate controls sampling and DWELL is not accessed. Default for DWELL is 1. The DWELL argument length is one word.	

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

Table 4–5 (Cont.) Subroutine Argument Usage

Argument	Meaning
IEFN	<p>Specifies the event flag number or completion routine address. The selected event flag is set at the end of each buffer transaction. If IEFN is 0 (default), event flag 22 is used.</p> <p>IEFN can also specify the address of a completion routine. This routine is called by the buffer management routine when a buffer is available and when the request is terminated, either successfully or with an error. The standard VMS calling and return sequences are used. The completion routine is called from an AST routine and is therefore at AST level.</p> <p>If IEFN specifies the address of a completion routine, the program must call the LPA\$IGTBUF subroutine to obtain the next buffer. If IEFN specifies an event flag, the program must call the LPA\$IWTBUF subroutine to obtain the next buffer and must use the %VAL operator:</p> <pre style="margin-left: 2em;">,%VAL(3), (Event flag 3) ,BFRFULL, (Address of completion routine)</pre> <p>The IEFN argument length is one longword.</p> <p>If multiple sweeps are initiated, they must use different event flags. The software does not enforce this policy.</p> <p>Event flag 23 is reserved for use by the LPA\$CLOCKA and LPA\$CLOCKB subroutines. If either of these subroutines is included in the user program, event flag 23 cannot be used. Also, if IEFN is defaulted, event flag 22 cannot be used in the user program.</p>
LDELAY	<p>Specifies the delay, in IRATE units, from the start event until the first sample is taken. The maximum value is 65,535; default is 1. The LDELAY argument length is one word. The LPA11-K supports the LDELAY argument in multirequest mode only.</p>
ICHN	<p>Specifies the number of the first I/O channel to be sampled. Default is channel 0. The ICHN argument length is one byte. The channel number is not the same as the channel assigned to the device by the \$ASSIGN system service. The LPA11-K uses the channel number to specify the multiplexer address of an A/D, D/A, or digital I/O device on the LPA11-K internal I/O bus.</p>
NCHN	<p>Specifies the number of I/O device channels to sample in a sample sequence. Default is 1. If the NCHN argument is 1, the single channel bit is set in the mode word of the start request descriptor array (RDA) when the sweep is started. The RDA contains the information needed by the LPA11-K for each command (see the <i>LPA 11-K Laboratory Peripheral Accelerator User's Guide</i>). The NCHN argument length is one word.</p>
IND	<p>Receives the VMS success or failure code of the call. The IND argument length is one longword.</p>

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

4.5.2 LPA\$ADSWP — Initiate Synchronous A/D Sampling Sweep

The LPA\$ADSWP subroutine initiates A/D sampling through an AD11-K.

The format of the LPA\$ADSWP subroutine call is as follows:

```
CALL LPA$ADSWP (IBUF,LBUF,[NBUF],[MODE],[DWELL],[IEFN],-  
               [LDELAY],[ICHN],[NCHN],[IND])
```

Arguments are as described in Section 4.5.1.2, with the following additions:

MODE Specifies sampling options. The VMS operating system defines the following sampling option values:

Value	Meaning
32	Parallel A/D conversion sample algorithm is used if dual A/D converters are specified (value = 8192). Absence of this bit implies the serial A/D conversion sample algorithm.
64	Multirequest mode request. Absence of this bit implies a dedicated mode request.
512	External trigger (Schmidt trigger 1). Dedicated mode only. This value is used when a user-supplied external sweep trigger is desired. The external trigger is supplied by the KW11-K (Schmidt trigger 1 output) to the AD11-K (external start input). If MODE=512, the user process must specify a Clock A rate of -1 for proper A/D sampling. This is nonclock-driven sampling (see Section 4.5.10). (The <i>LPA11-K Laboratory Peripheral Accelerator User's Guide</i> provides additional information on the use of external triggers.)
1024	Time stamped sampling with Clock B. The double word consists of one data word followed by the value of the LPA11-K's internal 16-bit counter at the time of the sample (see Section 2.4.3 in the <i>LPA11-K Laboratory Peripheral Accelerator User's Guide</i>). Multirequest mode only.
2048	Event marking. Multirequest mode only. (The <i>LPA11-K Laboratory Peripheral Accelerator User's Guide</i> describes event marking.)
4096	Start method. If selected, the digital input start method is used. If not selected, the immediate start method is used. Multirequest mode only.
8192	Dual A/D converters are to be used. Dedicated mode only.
16384	Buffer overrun is a nonfatal error. The LPA11-K will automatically default to fill buffer 0 if a buffer overrun condition occurs.

If MODE is defaulted, A/D sampling starts immediately with absolute channel addressing in dedicated mode. The LPA11-K does not support delays in dedicated mode.

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

IND Returns the success or failure status as follows:

0 = Error in call. Possible causes are the following: LPA\$SETIBF subroutine was not previously called; LPA\$RLSBUF subroutine was not previously called; size of data buffers disagrees with the size computed by the LPA\$SETIBF subroutine call.

1 = successful sweep started

nnn = VMS status code

4.5.3 LPA\$DASWP — Initiate Synchronous D/A Sweep

The LPA\$DASWP subroutine initiates D/A output to an AA11-K.

The format for the LPA\$DASWP subroutine call is as follows:

```
CALL LPA$DASWP (IBUF,LBUF,[NBUF],[MODE],[DWELL],[IEFN],-
               [LDELAY],[ICHN],[NCHN],[IND])
```

Arguments are as described in Section 4.5.1.2, with the following additions:

MODE Specifies the sampling options. The VMS operating system defines the following start criteria values:

Value	Meaning
0	Immediate start. This is the default value for MODE.
64	Multirequest mode. If not selected, this request is for dedicated mode.
4096	Start method. If selected, the digital input start method is used. If not selected, the immediate start method is used. Multirequest mode only.
16384	Buffer overrun is a nonfatal error. The LPA11-K will automatically default to empty buffer 0 if a buffer overrun condition occurs.

IND Returns the success or failure status as follows:

0 = Error in call. Possible causes are the following: LPA\$SETIBF subroutine was not previously called; LPA\$RLSBUF subroutine was not previously called; size of data buffers disagrees with the size computed by the LPA\$SETIBF subroutine call.

1 = successful sweep started

nnn = VMS status code

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

4.5.4 LPA\$DISWP — Initiate Synchronous Digital Input Sweep

The LPA\$DISWP subroutine initiates digital input through a DR11-K. It is applicable in multirequest mode only.

The format of the LPA\$DISWP subroutine call is as follows:

```
CALL LPA$DISWP (IBUF,LBUF,[NBUF],[MODE],[DWELL],[IEFN],-  
               [LDELAY],[ICHN],[NCHN],[IND])
```

Arguments are as described in Section 4.5.1.2, with the following additions:

MODE Specifies sampling options. The VMS operating system defines the following sampling option values:

Value	Meaning
0	Immediate start. This is the default value for MODE.
512	External trigger for DR11-K. (The <i>LPA 11-K Laboratory Peripheral Accelerator User's Guide</i> describes the use of external triggers.)
1024	Time stamped sampling with Clock B. The double word consists of one data word followed by the value of the internal LPA11-K 16-bit counter at the time of the sample (see Section 2.4.3 in the <i>LPA 11-K Laboratory Peripheral Accelerator User's Guide</i>).
2048	Event marking. (The <i>LPA 11-K Laboratory Peripheral Accelerator User's Guide</i> describes event marking.)
4096	Start method. If selected, the start method is digital input. If not selected, the start method is immediate. Multirequest mode only.
16384	Buffer overrun is a nonfatal error. The LPA11-K will automatically default to fill buffer 0 if a buffer overrun condition occurs.

IND Returns the success or failure status as follows:

0 = Error in call. Possible causes are the following: LPA\$SETIBF subroutine was not previously called; LPA\$RLSBUF subroutine was not previously called; size of data buffers disagrees with the size computed by the LPA\$SETIBF subroutine call.

1 = successful sweep started

nnn = VMS status code

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

4.5.5 LPA\$DOSWP — Initiate Synchronous Digital Output Sweep

The LPA\$DOSWP subroutine initiates digital output through a DR11-K. It is applicable in multirequest mode only.

The format of the LPA\$DOSWP subroutine call is as follows:

```
CALL LPA$DOSWP (IBUF,LBUF,[NBUF],[MODE],[DWELL],[IEFN],-
               [LDELAY],[ICHN],[NCHN],[IND])
```

Arguments are as described in Section 4.5.1.2, with the following additions:

MODE Specifies the sampling options. The VMS operating system defines the following values:

Value	Meaning
0	Immediate start. This is the default value for MODE.
512	External trigger for DR11-K. (The <i>LPA11-K Laboratory Peripheral Accelerator User's Guide</i> describes the use of external triggers.)
4096	Start method. If selected, digital input start. If not selected, immediate start.
16384	Buffer overrun is a nonfatal error. The LPA11-K will automatically default to empty buffer 0 if a buffer overrun condition occurs.

IND Returns the success or failure status as follows:

0 = Error in call. Possible causes are the following: LPA\$SETIBF subroutine was not previously called; LPA\$RLSBUF subroutine was not previously called; size of data buffers disagrees with the size computed by the LPA\$SETIBF subroutine call.

1 = successful sweep started

nnn = VMS status code

4.5.6 LPA\$LAMSKS — Set LPA11-K Masks and NUM Buffer

The LPA\$LAMSKS subroutine initializes a user buffer that contains a number to append to the logical name LPA11\$, a digital start word mask, an event mark mask, and channel numbers for the two masks.

The LPA\$LAMSKS subroutine must be called in the following cases:

- If users intend to use digital input starting or event marking
- If users do not want to use the default of LAA0 assigned to LPA11\$0
- If multiple LPA11-Ks are used

The format of the LPA\$LAMSKS subroutine call is as follows:

```
CALL LPA$LAMSKS (LAMSKB,[NUM],[IUNIT],[IDSC],[IEMC],[IDSW],-
               [IEMW],[IND])
```

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

Argument descriptions are as follows:

LAMSKB	Specifies a four-word array.
NUM	Specifies the number appended to LPA11\$. The sweep is started on the LPA11-K assigned to LPA11\$num.
IUNIT	Not used. This argument is present for compatibility only.
IDSC	Specifies the digital START word channel. Range is 0 through 4. The IDSC argument length is one byte.
IEMC	Specifies the event MARK word channel. Range is 0 through 4. The IEMC argument length is one byte.
IDSW	Specifies the digital START word mask. The IDSW argument length is one word.
IEMW	Specifies the event MARK word mask. The IEMW argument length is one word.
IND	Always equal to 1 (success). This argument is present for compatibility only.

4.5.7 **LPA\$SETADC — Set Channel Information for Sweeps**

The LPA\$SETADC subroutine establishes channel start and increment information for the sweep control subroutines (see Table 4-4). It must be called to initialize IBUF before the LPA\$SETADC subroutine is called.

The LPA\$SETADC subroutine can be called in either of the following formats:

```
CALL LPA$SETADC (IBUF,[IFLAG],[ICHN],[NCHN],[INC],[IND])
```

or

```
IND=LPA$SETADC (IBUF,[IFLAG],[ICHN],[NCHN],[INC])
```

Argument descriptions are as follows:

IND	Returns the success or failure status as follows: 0 = LPA\$SETIBF was not called prior to the LPA\$SETADC call 1 = LPA\$SETADC call successful
IBUF	The IBUF array specified in the LPA\$SETIBF call.
IFLAG	Reserved. This argument is present for compatibility only.
ICHN	Specifies the first channel number. Range is 0 through 255; default is 0. The ICHN argument length is one longword. If INC = 0, ICHN is the address of a random channel list. This address must be word aligned.
NCHN	Specifies the number of samples taken per sample sequence. Default is 1.
INC	Specifies the channel increment. Default is 1. If INC is 0, ICHN is the address of a random channel list. The INC argument length is one longword.

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

4.5.8 **LPA\$SETIBF** — Set IBUF Array for Sweeps

The LPA\$SETIBF subroutine initializes the IBUF array for use with the following subroutines:

LPA\$ADSWP	LPA\$DASWP	LPA\$DISWP
LPA\$DOSWP	LPA\$IBFSTS	LPA\$IGTBUF
LPA\$INXTBF	LPA\$IWTBUF	LPA\$RLSBUF
LPA\$RMVBUF	LPA\$SETADC	LPA\$STPSWP

The format of the LPA\$SETIBF subroutine call is as follows:

```
CALL LPA$SETIBF (IBUF,[IND],[LAMSKB],BUFO,[BUF1,...,BUF7])
```

Arguments are as described in Section 4.5.1.2, with the following additions:

IBUF	Specifies a 50-longword array that is initialized by this subroutine. IBUF must be longword-aligned. (See Table 4–5 for additional information on IBUF.)
IND	Returns the success or failure status as follows: 0 = Error in call. Possible causes are the following: incorrect number of arguments; IBUF array not longword-aligned; buffer addresses not equidistant. 1 = IBUF initialized successfully
LAMSKB	Specifies the name of a four-word array. This array allows the use of multiple LPA11-Ks within the same program because the argument used to start the sweep is specified by the LPA\$LAMSKS subroutine call. (See Section 4.5.6 for a description of the LPA\$LAMSKS subroutine.)
BUFO, . . .	Specify the names of the buffers. A maximum of eight buffers can be specified. At least two buffers must be specified to provide continuous sampling. The LPA11-K driver requires that all buffers be contiguous. To ensure this, the LPA\$SETIBF subroutine verifies that all buffer addresses are equidistant. Buffers must be longword-aligned.

4.5.9 **LPA\$STPSWP** — Stop In-Progress Sweep

The LPA\$STPSWP subroutine allows you to stop a sweep that is in progress.

The format of the LPA\$STPSWP subroutine call is as follows:

```
CALL LPA$STPSWP (IBUF,[IWHEN],[IND])
```

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

Arguments are as described in Section 4.5.1.2, with the following additions:

IBUF	The IBUF array specified in the LPA\$ADSWP, LPA\$DASWP, LPA\$DISWP, or LPA\$DOSWP subroutine call that initiated the sweep.
IWHEN	Specifies when to stop the sweep. The VMS operating system defines the following values: 0 = Abort sweep immediately. Uses the \$CANCEL system service. This is the default sweep stop. 1 = Stop sweep when the current buffer transaction is completed. (This is the preferred way to stop requests.)
IND	Receives a success or failure code in the standard VMS format: 1 = Success nnn = VMS error code issued by the \$CANCEL system service

Note that, when the LPA\$STPSWP subroutine is returned, the sweep cannot be stopped. If it is necessary to wait until the sweep has stopped, you can call the LPA\$IWTBUF subroutine in a loop until it returns IBUFNO = -1 (see Section 4.5.16).

4.5.10 LPA\$CLOCKA — Clock A Control

The LPA\$CLOCKA subroutine sets the clock rate for Clock A.

The format of the LPA\$CLOCKA subroutine call is as follows:

```
CALL LPA$CLOCKA (IRATE,IPRSET,[IND],[NUM])
```

Arguments are as described in Section 4.5.1.2, with the following additions:

IRATE Specifies the clock rate. One of the following values must be specified:

Value	Meaning
-1	Direct-coupled Schmidt trigger 1. Used only for A/D sweeps in dedicated mode, that is, MODE = 512 (see Section 4.5.2).
0	Clock B overflow or no rate
1	1 MHz
2	100 kHz
3	10 kHz
4	1 kHz
5	100 Hz
6	Schmidt trigger 1
7	Line frequency

IPRSET Specifies the clock preset value. Maximum of 16 bits. The LPA\$XRATE subroutine can be used to calculate this value. The clock rate divided by the clock preset value yields the clock overflow rate.

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

IND Receives a success or failure code as follows:
 1 = Clock A set successfully
 nnn = VMS error code indicating an I/O error

NUM Specifies the number to be appended to the logical name
 LPA11\$. The default value is 0. This subroutine sets Clock A
 on the LPA11-K assigned to LPA11\$num.

4.5.11 LPA\$CLOCKB — Clock B Control

The LPA\$CLOCKB subroutine provides the user with control of the KW11-K Clock B.

The format of the LPA\$CLOCKB subroutine call is as follows:

CALL LPA\$CLOCKB ([IRATE],IPRSET,MODE,[IND],[NUM])

Arguments are as described in Section 4.5.1.2, with the following additions:

IRATE Specifies the clock rate. One of the following values must be specified:

Value	Meaning
0	Stops Clock B
1	1 MHz
2	100 kHz
3	10 kHz
4	1 kHz
5	100 Hz
6	Schmidt trigger 3
7	Line frequency

If IRATE is 0 (default), the clock is stopped and the IPRSET and MODE arguments are ignored.

IPRSET Specifies the preset value by which the clock rate is divided to yield the overflow rate. Maximum of eight bits. Overflow events can be used to drive Clock A. The LPA\$XRATE subroutine can be used to calculate the IPRSET value.

MODE Specifies options. The VMS operating system defines the following values:

- 1 = Clock B operates in noninterrupt mode.
- 2 = The feed B to A bit in the Clock B status register will be set (see Section 3.3 of the *LPA11-K Laboratory Peripheral Accelerator User's Guide*).

IND Receives a success or failure code as follows:
 1 = Clock B set successfully
 nnn = VMS error code indicating an I/O error

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

NUM Specifies the number to be appended to the logical name LPA11\$. The default value is 0. This subroutine sets Clock B on the LPA11-K assigned to LPA11\$num.

4.5.12 LPA\$XRATE — Compute Clock Rate and Preset Value

The LPA\$XRATE subroutine computes the clock rate and preset value for the LPA\$CLOCKA and LPA\$CLOCKB subroutines using the specified intersample interval (AINTRVL).

The LPA\$XRATE subroutine can be called in either of the following formats:

```
CALL LPA$XRATE (AINTRVL,IRATE,IPRSET,IFLAG)
```

or

```
ACTUAL=LPA$XRATE(AINTRVL,IRATE,IPRSET,IFLAG)
```

Arguments are as described in Section 4.5.1.2, with the following additions:

AINTRVL	Specifies the intersample time selected by the user. The time is expressed in decimal seconds. Data type is floating point.
IRATE	Receives the computed clock rate as a value from 1 through 5.
IPRSET	Receives the computed clock preset value.
IFLAG	If the computation is for Clock A, IFLAG is 0; if for Clock B, IFLAG is not 0 (the maximum preset value is 255). The IFLAG argument length is one byte.
ACTUAL	Receives the actual intersample time if called as a function. Data type is floating point. If there are truncation and round-off errors, the resulting intersample time can be different from the specified intersample time. Note that when the LPA\$XRATE subroutine is called from VAX FORTRAN IV-PLUS programs as a function, it must be explicitly declared a real function. Otherwise, the LPA\$XRATE subroutine defaults to an integer function.

If AINTRVL is either too large or too small to be achieved, both IRATE and ACTUAL are returned to 0.

4.5.13 LPA\$IBFSTS — Return Buffer Status

The LPA\$IBFSTS subroutine returns information on the buffers used in a sweep.

The format of the LPA\$IBFSTS subroutine call is as follows:

```
CALL LPA$IBFSTS (IBUF,ISTAT)
```

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

Argument descriptions are as follows:

IBUF	The IBUF array specified in the call that initiated the sweep.
ISTAT	Specifies a longword array with as many elements as there are buffers involved in the sweep (maximum of eight). LPA\$IBFSTS fills each array element with the status of the corresponding buffer: +2 = Buffer in device queue. LPA\$RLSBUF has been called for this buffer. +1 = Buffer in user queue. The LPA11-K has filled (data input) or emptied (data output) this buffer. 0 = Buffer is not in any queue. -1 = Buffer is in the in-use queue; that is, it is either being filled or emptied, or it is the next to be filled or emptied by the LPA11-K.

4.5.14 LPA\$IGTBUF — Return Buffer Number

The LPA\$IGTBUF subroutine returns the number of the next buffer to be processed by the application program, the buffer at the head of the user queue (see Figure 4-3). It should be called by a completion routine at AST level to determine the next buffer to process. If an event flag was specified in the start sweep call, the LPA\$IWTBUF, not the LPA\$IGTBUF subroutine, should be called.

The LPA\$IGTBUF subroutine can be called in either of the following formats:

```
CALL LPA$IGTBUF (IBUF,IBUFNO)
```

```
IBUFNO=LPA$IGTBUF(IBUF)
```

Arguments are as described in Section 4.5.1.2, with the following additions:

IBUF	The IBUF array specified in the call that initiated the sweep.
IBUFNO	Returns the number of the next buffer to be filled or emptied by the application program.

Table 4-6 lists the possible combinations of IBUFNO and IOSB contents on the return from a call to the LPA\$IGTBUF subroutine. The first four words of the IBUF array contain the I/O status block (IOSB). If IBUFNO is -1, the IOSB must be checked to determine the reason.

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

Table 4–6 LPA\$IGTBUF Call — IBUFNO and IOSB Contents

IBUFNO	IOSB(1)	IOSB(2)	IOSB(3),(4)	Meaning
n	0	(byte count)	0	Normal buffer complete.
-1	0	0	0	No buffers in queue. Request still active.
-1	1	0	0	No buffers in queue. Sweep terminated normally.
-1	VMS error code	0	LPA11-K ready-out and maintenance registers (only if SS\$DEVREQERR, SS\$_CTRLERR, or SS\$DEVCMDERR is returned)	No buffers in queue. Sweep terminated due to error condition. Section 4.6 describes the VMS error codes; Appendix A of the <i>LPA 11-K Laboratory Peripheral Accelerator User's Guide</i> lists the LPA11-K error codes.

4.5.15 LPA\$INXTBF — Set Next Buffer to Use

The LPA\$INXTBF subroutine alters the normal buffer selection algorithm so that you can specify the next buffer to be filled or emptied. The specified buffer is reinserted at the head of the device queue.

The LPA\$INXTBF subroutine can be called in either of the following formats:

```
CALL LPA$INXTBF (IBUF,IBUFNO,IND)
```

```
IND=LPA$INXTBF(IBUF,IBUFNO)
```

Arguments are as described in Section 4.5.1.2, with the following additions:

IBUF	The IBUF array specified in the call that initiated the sweep.
IBUFNO	Specifies the number of the next buffer to be filled or emptied. The buffer must already be in the device queue.
IND	Returns the result of the call as follows: 0 = Specified buffer not in the device queue 1 = Next buffer successfully set

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

4.5.16 LPA\$IWTBUF — Return Next Buffer or Wait

The LPA\$IWTBUF subroutine returns the next buffer to be processed by the application program, the buffer at the head of the user queue. If the user queue is empty, the LPA\$IWTBUF subroutine waits until a buffer is available. If a completion routine was specified in the call that initiated the sweep, the LPA\$IGTBUF, not the LPA\$IWTBUF subroutine, should be called.

The LPA\$IWTBUF subroutine can be called in either of the following formats:

```
CALL LPA$IWTBUF (IBUF,[IEFN],IBUFNO)
IBUFNO=LPA$IWTBUF(IBUF,[IEFN])
```

Arguments are as described in Section 4.5.1.2, with the following additions:

- IBUF The IBUF array specified in the call that initiated the sweep.
- IEFN Not used. This argument is present to provide compatibility with the operating system. (The event flag is the one specified in the start sweep call.)
- IBUFNO Returns the number of the next buffer to be filled or emptied by the application program.

Table 4-7 lists the possible combinations of IBUFNO and I/O status block contents on the return from a call to the LPA\$IWTBUF subroutine. The first four words of the IBUF array contain the I/O status block. If IBUFNO is -1, the I/O status block must be checked to determine the reason.

Table 4-7 LPA\$IWTBUF Call — IBUFNO and IOSB Contents

IBUFNO	IOSB(1)	IOSB(2)	IOSB(3),(4)	Meaning
n	0	(byte count)	0	Normal buffer complete.
-1	1	0	0	No buffers in queue. Sweep terminated normally.
-1	VMS error code	0	LPA11-K ready-out and maintenance registers (only if SS\$_DEVREQERR, SS\$_CTRLERR, or SS\$_DEVCMDErr is returned)	No buffers in queue. Sweep terminated due to error condition. Section 4.6 describes the VMS error codes; Appendix A of the <i>LPA11-K Laboratory Peripheral Accelerator User's Guide</i> lists the LPA11-K error codes.

4.5.17 LPA\$RLSBUF — Release Data Buffer

The LPA\$RLSBUF subroutine declares one or more buffers available to be filled or emptied by the LPA11-K. It inserts the buffer at the tail of the device queue (see Figure 4-3).

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

The format of the LPA\$RLSBUF subroutine call is as follows:

```
CALL LPA$RLSBUF (IBUF,[IND],INDEX0,INDEX1,...,INDEXN)
```

Arguments are as described in Section 4.5.1.2, with the following additions:

IBUF	The IBUF array specified in the call that initiated the sweep.
IND	Returns the success or failure status as follows: 0 = Buffer number was illegal, the number of arguments specified was incomplete, or a double buffer overrun occurred. A double buffer overrun can occur only if buffer overrun was specified as a nonfatal error, a buffer overrun occurs, and buffer 0 was not released (probably on the user queue after a previous buffer overrun). 1 = Buffer(s) released successfully.
INDEX0, . . .	Specify the indexes (0-7) of the buffers to be released. A maximum of eight indexes can be specified.

The LPA\$RLSBUF subroutine must be called to release a buffer (or buffers) to the device queue before the sweep is initiated. (See Section 4.5.1.1 for a discussion of buffer management.) Note that the LPA\$RLSBUF subroutine does not verify whether the specified buffers are already in a queue. If a buffer is released when it is already in a queue, the queue pointers are invalidated and unpredictable results can occur.

If buffer overrun is specified as a nonfatal error, buffer 0 should not be released before the sweep is initiated. However, if either the LPA\$IGTBUF or LPA\$IWTBUF subroutine returns buffer 0, it should be released. In this case, buffer 0 is set aside (not placed on a queue) until the buffer overrun occurs. If a buffer overrun occurs and buffer 0 was not released, the LPA\$RLSBUF subroutine returns an error the next time buffer 0 is released.

4.5.18 LPA\$RMVBUF — Remove Buffer from Device Queue

The LPA\$RMVBUF subroutine removes a buffer from the device queue.

The format of the LPA\$RMVBUF subroutine call is as follows:

```
CALL LPA$RMVBUF (IBUF,IBUFNO,[IND])
```

Arguments are as described in Section 4.5.1.2, with the following additions:

IBUF	The IBUF array specified in the call that initiated the sweep.
IBUFNO	Specifies the number of the buffer to remove from the device queue.
IND	Returns the success or failure status as follows: 0 = Buffer not found in the device queue 1 = Buffer successfully removed from the device queue

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

4.5.19 **LPA\$CVADF — Convert A/D Input to Floating-Point**

The LPA\$CVADF subroutine converts A/D input values to floating-point numbers. It is supported to provide compatibility with the VMS operating system.

The LPA\$CVADF subroutine can be called in either of the following formats:

```
CALL LPA$CVADF (IVAL,VAL)
```

```
VAL=LPA$CVADF(IVAL)
```

Argument descriptions are as follows:

IVAL Contains the value (bits 11:0) read from the A/D input. Bits 15:12 are 0.

VAL Receives the floating-point value.

4.5.20 **LPA\$FLT16 — Convert Unsigned 16-bit Integer to Floating-Point**

The LPA\$FLP16 subroutine converts unsigned 16-bit integers to floating point. It is supported to provide compatibility with the VMS operating system.

The LPA\$FLT16 subroutine can be called in either of the following formats:

```
CALL LPA$FLT16 (IVAL,VAL)
```

```
VAL=LPA$FLT16(IVAL)
```

Argument descriptions are as follows:

IVAL An unsigned 16-bit integer.

VAL Receives the converted value.

4.5.21 **LPA\$LOADMC — Load Microcode and Initialize LPA11-K**

The LPA\$LOADMC subroutine provides a program interface to the LPA11-K microcode loader. It sends a load request through a mailbox to the loader process to load microcode and to initialize an LPA11-K. (Section 4.7.1 describes the microcode loader process.)

The format of the LPA\$LOADMC subroutine call is as follows:

```
CALL LPA$LOADMC (([ITYPE],[NUM],[IND],[IERROR])
```

Laboratory Peripheral Accelerator Driver

4.5 High-Level Language Interface

Argument descriptions are as follows:

ITYPE	The type of microcode to be loaded. The VMS operating system defines the following values:								
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Value</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Multirequest mode; default value</td> </tr> <tr> <td>2</td> <td>Dedicated A/D mode</td> </tr> <tr> <td>3</td> <td>Dedicated D/A mode</td> </tr> </tbody> </table>	Value	Meaning	1	Multirequest mode; default value	2	Dedicated A/D mode	3	Dedicated D/A mode
Value	Meaning								
1	Multirequest mode; default value								
2	Dedicated A/D mode								
3	Dedicated D/A mode								
NUM	The number to be appended to the logical name LPA11\$. The default value is 0.								
IND	Receives the completion status as follows: 1 = Microcode loaded successfully nnn = VMS error code								
IERROR	Provides additional error information. Receives the second longword of the I/O status block if SS\$_CTRLERR, SS\$_DEVCMDEERR, or SS\$_DEVREQERR is returned in IND. Otherwise, the contents of IERROR are undefined.								

4.6 I/O Status Block

The I/O status block (IOSB) format for the load microcode, start microprocessor, initialize LPA11-K, set clock, and start data transfer request QIO functions is shown in Figure 4-4.

Figure 4-4 I/O Functions IOSB Content

31	0
16	15
byte count	status
LPA11-K maintenance status	LPA11-K ready-out

ZK-662-82

VMS status values and the byte count are returned in the first longword. Status values are defined by the \$SSDEF macro. The byte count is the number of bytes transferred by a IO\$_LOADMCODE request. If SS\$_CTRLERR, SS\$_DEVCMDEERR, or SS\$_DEVREQERR is returned in the status word, the second longword contains the LPA11-K ready-out register and LPA11-K maintenance status register values present at the completion of the request. The high byte of the ready-out register contains the specific LPA11-K error code (see Appendix A of the *LPA11-K Laboratory Peripheral Accelerator User's Guide*). Appendix A of this manual lists the status returns for LPA11-K I/O functions. (The *VMS System Messages and Recovery Procedures Reference Volume* provides explanations and suggested user actions for these returns.)

Laboratory Peripheral Accelerator Driver

4.6 I/O Status Block

If high-level language library procedures are used, the status returns listed in Appendix A can be returned from the resultant QIO functions. Since buffers are filled by these procedures asynchronously, two I/O status blocks are provided in the IBUF array: one for the high-level language procedures and one for the LPA11-K driver. The first four words of the IBUF array contain the I/O status block for the high-level language procedures.

4.7 Loading LPA11-K Microcode

The microcode loading and device initialization routines automatically load microcode during system initialization (if specified in the system manager's startup file) and during power recovery. These routines also allow a nonprivileged user to load microcode and to restart the system.

The LPA11-K loader and initialization routines consist of the following parts:

- A microcode loader process that loads any of the three microcode versions, initializes the LPA11-K, and sets the clock rate. Loading is initiated by either a mailbox request or a power recovery AST. This process requires permanent mailbox (PRMMBX) and physical I/O privileges.
- An operator process that accepts operator commands or indirect file commands to load microcode and to initialize an LPA11-K. This process uses a mailbox to send a load request to the loader process; temporary mailbox (TMPMBX) privilege is required.
- An LPA11-K procedure library routine that provides a program interface to the LPA11-K microcode loader. The procedure sends a load request through a mailbox to the loader process to load microcode and to initialize an LPA11-K. Section 4.5.21 describes that routine in greater detail.

4.7.1 Microcode Loader Process

The microcode loader process loads microcode, initializes a specific LPA11-K, and sets the clock at the default rate (10 kHz interrupt rate). A bit set in a controller bit map indicates that the specified controller was loaded. The process specifies a power recovery AST, creates a mailbox whose name (LPA\$LOADER) is entered in the system logical name table, and then hibernates.

The correct device configuration is determined automatically. When LPA11-K initialization is performed, every possible device (see Table 4-1) is specified as present on the LPA11-K. If the LPA11-K returns a "device not found" error, the LPA11-K is reinitialized with that device omitted.

On receipt of a power recovery AST, the loader process examines the controller bit map to determine which LPA11-Ks have been loaded. For each LPA11-K, the loader process performs the following functions:

- Obtains device characteristics
- Reloads the microcode previously loaded
- Reinitializes the LPA11-K
- Sets Clock A to the previous rate and preset value

Laboratory Peripheral Accelerator Driver

4.7 Loading LPA11-K Microcode

4.7.2 Operator Process

The operator process loads microcode and initializes an LPA11-K through either terminal or indirect file commands. To run the operator process, type RUN SYS\$SYSTEM:LALOAD. The command input syntax is as follows:

`devname/type`

devname is the device name of the LPA11-K to be loaded. A logical name can be specified. However, only one level of logical name translation is performed. If *devname* is omitted, LAA0 is the default name. If */type* appears, it specifies one of the following types of microcode to load:

- `/MULTI_REQUEST`—Multirequest mode
- `/ANALOG_DIGITAL`—Dedicated A/D mode
- `/DIGITAL_ANALOG`—Dedicated D/A mode

If */type* type is omitted, `/MULTI_REQUEST` is the default.

After receiving the command, the operator process formats a message and sends it to the loader process. Completion status is returned through a return mailbox.

4.8 RSX-11M/M-PLUS and VMS Differences

This section lists those areas of the VMS high-level language support routines that differ from the RSX-11M LPA11-K routines. The *RSX-11M-PLUS and Micro/R SX I/O Drivers Manual* provides a detailed description of the RSX-11M LPA11-K support routines. Differences between the VMS and RSX-11M/M-PLUS routines can be determined by comparing the descriptions in the *RSX-11M-PLUS and Micro/R SX I/O Drivers Manual* with the descriptions for the VMS routines in the preceding sections of this chapter.

4.8.1 General

The following are general features of VMS high-level support routines:

- The LUN argument is not used. The NUM argument specifies the number to be appended to the logical name LPA11\$.
- All routine names have the prefix LPA\$.
- In the LPA\$SETIBF routine, buffer addresses are checked for contiguity.
- In the LPA\$LAMSKS routine, the IUNIT argument is not used.
- In the LPA\$IWTBUF routine, the IEFN argument is not used. The event flag specified in the sweep routine is used.
- The combinations of IBUFNO and I/O status block (IOSB) values returned by the LPA\$IWTBUF and LPA\$IGTBUF subroutines are different.

Laboratory Peripheral Accelerator Driver

4.8 RSX-11M/M-PLUS and VMS Differences

4.8.2 Alignment and Length

The following are features of alignment and length in VMS high-level support routines:

- Buffers must be contiguous.
- Buffers must be longword-aligned.
- The random channel list (RCL) must be word-aligned.
- The IBUF array length is 50 longwords and must be longword-aligned.

4.8.3 Status Returns

The following are features of status returns in VMS high-level support routines:

- The I/O status block (IOSB) length is eight bytes; numeric values of errors differ.
- Several routines return the following:
 - 1 = Success
 - 0 = Failure detected in support routine
 - nnn = VMS status code; failure detected in system service

4.8.4 Sweep Routines

The following are features of sweep routines in VMS high-level support routines:

- If an event flag is specified, it must be within a %VAL() construction.
- A tenth argument, IND, is added to return the success or failure status.

4.9 Programming Examples

The following programming examples use LPA11-K high-level language procedures and LPA11-K Queue I/O functions.

The *VMS Device Support Manual* volume contains information that is applicable to LPA11-K programming.

Laboratory Peripheral Accelerator Driver

4.9 Programming Examples

4.9.1 LPA11-K High-Level Language Program (Program A)

This sample program (Example 4-1) is an example of how the LPA11-K high-level language procedures perform an A/D sweep using three buffers. The program uses default arguments whenever possible to illustrate the simplest possible calls. The program assumes that dedicated mode microcode has previously been loaded into the LPA11-K. Table 4-8 lists the variables used in this program.

Table 4-8 Program A Variables

Variable	Description
BUFFER	The data buffer array. BUFFER is a common area to guarantee longword alignment.
IBUF	The LPA 11-K high-level language procedures use the IBUF array for local storage.
BUFNUM	BUFNUM contains the buffer number returned by LPA\$IWTFBUF. In this example, the possible values are 0, 1, and 2.
ISTAT	ISTAT contains the status return from the high-level language calls.

Example 4-1 LPA11-K High-Level Language Program (Program A)

```
C *****
C
C                               PROGRAM A
C
C *****

      INTEGER*2    BUFFER(1000,0:2),IOSB(4)
      INTEGER*4    IBUF(50),ISTAT,BUFNUM

      COMMON/AREA1/BUFFER

      EQUIVALENCE    (IOSB(1),IBUF(1))

C
C Set clock rate to 1 khz, clock preset to -10.
C
      CALL LPA$CLOCKA(4,-10,ISTAT)
      IF (.NOT. ISTAT) GO TO 950

C
C Initialize IBUF array for sweep.
C
      CALL LPA$SETIBF(IBUF,ISTAT, ,BUFFER(1,0),BUFFER(1,1),BUFFER(1,2))
      IF (.NOT. ISTAT) GO TO 950

C
C Release all the buffers. Note use of buffer numbers rather than
C buffer names.
C
      CALL LPA$RLSBUF(IBUF,ISTAT,0,1,2)
      IF (.NOT. ISTAT) GO TO 950
```

Example 4-1 Cont'd. on next page

Laboratory Peripheral Accelerator Driver

4.9 Programming Examples

Example 4-1 (Cont.) LPA11-K High-Level Language Program (Program A)

```
C
C Start A/D sweep
C
          CALL LPA$ADSWP(IBUF,1000,50,,,,,ISTAT)
          IF (.NOT. ISTAT) GO TO 950
C
C Get next buffer filled with data. If BUFNUM is negative, there
C are no more buffers and the sweep is stopped.
C
100      BUFNUM = LPA$IWTBUF(IBUF)
          IF (BUFNUM .LT. 0) GO TO 800
C
C Process data in buffer (1,BUFNUM) to buffer (1000,BUFNUM).
.
.
.
(Application-dependent code is inserted at this point.)
.
.
C Release buffer is filled again.
C
200      CALL LPA$RLSBUF(IBUF,ISTAT,BUFNUM)
          IF (.NOT. ISTAT) GO TO 950
          GO TO 100
C
C There are no more buffers to process. Check to ensure that the
C sweep ended successfully. IOSB(1) contains either 1 or a
C VMS status code.
C
800      IF (.NOT. IOSB(1)) CALL LIB$STOP(%VAL(IOSB(1)))
          PRINT *, 'SUCCESSFUL COMPLETION'
          GO TO 2000
C
C Error return from subroutine. ISTAT contains either 0 or a
C VMS error code.
C
950      IF (ISTAT .NE. 0) CALL LIB$STOP(%VAL(ISTAT))
          PRINT *, 'ERROR IN LPA11-K SUBROUTINE CALL'
2000     STOP
          END
C *****
```

4.9.2 LPA11-K High-Level Language Program (Program B)

This program (Example 4-2) is a more complex example of LPA11-K operations performed by the LPA11-K high-level language procedures. The following operations are demonstrated:

- Program-requested loading of LPA11-K microcode
- Setting the clock at a specified rate
- Use of nondefault arguments whenever possible
- An A/D sweep that uses an event flag
- A D/A sweep that uses a completion routine

Laboratory Peripheral Accelerator Driver

4.9 Programming Examples

- Buffer overrun set (buffer overrun is a nonfatal error)
- Random channel list (RCL) addressing
- Sequential channel addressing

Table 4–9 lists the variables used in this program.

Table 4–9 Program B Variables

Variable	Description
AD	An array of buffers for ahA/D sweep (8 buffers of 500 words each)
DA	An array of buffers for a D/A sweep (2 buffers of 2000 words each)
IBUFAD	The IBUF array for an A/D sweep
IBUFDA	The IBUF array for a D/A sweep
RCL	The array that contains the random channel list (RCL)
ADIOSB	The array that contains the I/O status block for the A/D sweep. Equivalenced to the beginning of IBUFAD
DAIOSB	The array that contains the I/O status block for the D/A sweep. Equivalenced to the beginning of IBUFDA
ISTAT	Contains the status return from the high-level language calls

Example 4–2 LPA11-K High-Level Language Program (Program B)

```
C *****
C
C                               Program B
C
C *****
C
C      EXTERNAL FILLBF
C      REAL*4 LPA$XRATE
C
C      INTEGER*2 AD(500,0:7),DA(2000,0:1),RCL(5),MODE,IPRSET
C      INTEGER*2 ADIOSB(4),DAIOSB(4)
C
C      INTEGER*4 IBUFAD(50),IBUFDA(50),LAMSKB(2)
C      INTEGER*4 ISTAT,IERROR,IRATE,BUFNUM
C
C      REAL*4 PERIOD
C
C      COMMON /SWEEP/AD,DA,IBUFAD,IBUFDA
C
C      EQUIVALENCE (IBUFAD(1),ADIOSB(1)),(IBUFDA(1),DAIOSB(1))
C
C      PARAMETER MULTI=1, HBIT='8000'X, LSTCHN=HBIT+7
C
C      Set up random channel list. Note that the last word must have bit
C      15 set.
C
C      DATA RCL/2,6,3,4,LSTCHN/
```

Example 4–2 Cont'd. on next page

Laboratory Peripheral Accelerator Driver

4.9 Programming Examples

Example 4-2 (Cont.) LPA11-K High-Level Language Program (Program B)

```
C *****
C
C Load multirequest mode microcode and set the clock overflow rate
C to 5 khz.
C
C *****
C Load microcode on LPA11-K assigned to LPA11$3.
C
      CALL LPA$LOADMC(MULTI,3,ISTAT,IERROR)
      IF (.NOT. ISTAT) GO TO 5000
C
C Compute clock rate and preset. Set clock 'A' on LPA11-K
C assigned to LPA11$3.
C
      PERIOD = LPA$XRATE(.0002,IRATE,IPRSET,0)
      IF (PERIOD .EQ. 0.0) GO TO 5500
      CALL LPA$CLOCKA(IRATE,IPRSET,ISTAT,3)
      IF (.NOT. ISTAT) GO TO 5000
C *****
C
C Set up for A/D sweep
C
C *****
C Initialize IBUF array. Note the use of the LAMSKB argument because
C the LPA11-K assigned to LPA11$3 is used.
C
      CALL LPA$SETIBF(IBUFAD,ISTAT,LAMSKB,AD(1,0),AD(1,1),AD(1,2),
      1 AD(1,3),AD(1,4),AD(1,5),AD(1,6),AD(1,7))
      IF (.NOT. ISTAT) GO TO 5000
      CALL LPA$LAMSKS(LAMSKB,3)
C
C Set up random channel list sampling (20 samples in a sample
C sequence).
C
      CALL LPA$SETADC(IBUFAD,,RCL,20,0,ISTAT)
      IF (.NOT. ISTAT) GO TO 5000
C
C Release buffers for A/D sweep. Note that buffer 0 is not
C released because buffer overrun will be specified as nonfatal.
C
      CALL LPA$RLSBUF(IBUFAD,ISTAT,1,2,3,4,5,6,7)
      IF (.NOT. ISTAT) GO TO 5000
```

Example 4-2 Cont'd. on next page

Laboratory Peripheral Accelerator Driver

4.9 Programming Examples

Example 4-2 (Cont.) LPA11-K High-Level Language Program (Program B)

```
C
C Assume sweep should be stopped when the last sample in buffer
C equals 0. Note that the sweep actually stops when the buffer
C currently being filled is full. Also note that LPA$IWTBUF
C continues to be called until there are no more buffers to process.
C
      IF (AD(500,BUFNUM) .NE. 0) GO TO 200
      CALL LPA$STPSWP(IBUFAD,1,ISTAT)
      IF (.NOT. ISTAT) GO TO 5000

C
C After the data is processed, the buffer is released to be
C filled again. Then the next buffer is obtained from A/D.
C
200   CALL LPA$RLSBUF(IBUFAD,ISTAT,BUFNUM)
      IF (.NOT. ISTAT) GO TO 5000
      GO TO 100

C
C Enter here when A/D sweep has ended. Check for error or
C successful end. (Note: Assume that the D/A sweep has already
C ended - see completion routine FILLBF.)
C
1000  IF(ADIOSB(1)) GO TO 6000
      CALL LIB$STOP(%VAL(ADIOSB(1)))

C
C Enter here if there was an error returned from one of the
C LPA11-K high-level language calls. ISTAT contains either 0
C or a VMS status code.
C
5000  IF (ISTAT .NE. 0) CALL LIB$STOP (%VAL(ISTAT))
5500  PRINT *,'ERROR IN LPA11-K SUBROUTINE CALL'
      GO TO 7000

6000  PRINT *,'SUCCESSFUL COMPLETION'
7000  STOP
      END

C *****
C
C Subroutine FILLBF
C
C *****
C
C The FILLBF subroutine is called whenever the D/A has emptied a
C buffer, and that buffer is available to be refilled. This
C subroutine gets the buffer, fills it, and releases it back to the
C LPA11-K. Note that the D/A sweep is stopped automatically after
C 15 buffers have been filled. Also note that FILLBF is called by
C an AST handler. It is therefore called asynchronously from the
C main program at AST level. Care should be exercised when accessing
C variables that are common to both levels.
C
      INTEGER*2 AD(500,0:7),DA(2000,0:1),ADIOSB(4)
      INTEGER*4 IBUFAD(50),IBUFDA(50),BUFNUM,ISTAT
      EQUIVALENCE (IBUFDA(1),ADIOSB(1))
      COMMON /SWEEP/AD,DA,IBUFAD,IBUFDA
```

Example 4-2 Cont'd. on next page

Laboratory Peripheral Accelerator Driver

4.9 Programming Examples

Example 4-2 (Cont.) LPA11-K High-Level Language Program (Program B)

```

C
C Get buffer number of next buffer to fill.
C
          BUFNUM = LPA$IGTBUF(IBUFDA)
          IF (BUFNUM .LT. 0) GO TO 3000

C
C Fill buffer with data for output to D/A.
.
.
.
(Application-dependent code is inserted here to fill buffer
DA(1,BUFNUM) through DA(2000,BUFNUM) with data.)
.
.
.

C
C Release buffer
C
          CALL LPA$RLSBUF(IBUFDA, ISTAT, BUFNUM)
          GO TO 4000

C
C Check for successful end of sweep.
C
3000    IF(DAIOSE(1)) GO TO 4000

C
C Error in sweep
C
          CALL LIB$STOP(%VAL(DAIOSE(1)))

4000    RETURN
        END
C *****

```

4.9.3 LPA11-K QIO Functions Program (Program C)

This sample program (Example 4-3) uses QIO functions to start an A/D data transfer from an LPA11-K. (The program assumes multirequest mode microcode has been loaded.) Sequential channel addressing is used. The data transfer is stopped after 100 buffers have been filled; no action is taken with the data as the buffers are filled. Note that this program starts the data transfer and then waits until the QIO operation completes.

Laboratory Peripheral Accelerator Driver

4.9 Programming Examples

Example 4–3 LPA11-K QIO Functions Program (Program C)

```
*****  
;  
;  
;                               Program C  
;  
*****  
  
    .TITLE LPA11-K EXAMPLE PROGRAM  
    .IDENT /V01/  
  
    .PSECT LADATA, LONG  
  
IOSB:  .BLKQ  1           ; I/O status block  
COUNT:  .LONG  0        ; Count of buffers filled  
  
CBUFF:                                     ; Command buffer for start  
; Data QIO  
    .WORD  ^X20A          ; Mode = Sequential channel  
; Addressing, A/D,  
; multirequest mode  
    .WORD  3              ; Valid buffer mask  
; (4 buffers)  
    .LONG  USW             ; User Status Word address  
    .LONG  4000            ; Aggregate buffer length  
    .LONG  DATA_BUFFERO   ; Address of data buffers  
    .LONG  0               ; No random channel list  
; length  
    .LONG  0               ; No random channel list  
; address  
    .WORD  10              ; Delay  
    .BYTE  0               ; Start channel  
    .BYTE  1               ; Channel increment  
    .WORD  16              ; Number of samples in  
; sample sequence  
    .WORD  1               ; Dwell  
    .BYTE  0               ; Start word number  
    .BYTE  0               ; Event mark word  
    .WORD  0               ; Start word mask  
    .WORD  0               ; Event mark mask  
    .WORD  0               ; Fills out command buffer  
  
USW:    .WORD  0           ; User Status Word  
  
    .ALIGN LONG           ; Buffers must be  
; longword aligned  
DATA_BUFFERO: .BLKW 500   ; Data buffers  
DATA_BUFFER1: .BLKW 500  
DATA_BUFFER2: .BLKW 500  
DATA_BUFFER3: .BLKW 500
```

Example 4–3 Cont'd. on next page

Laboratory Peripheral Accelerator Driver

4.9 Programming Examples

Example 4-3 (Cont.) LPA11-K QIO Functions Program (Program C)

```

DEVNAME: .ASCID /LAAO/
CHANNEL: .BLKW 1 ; Contains channel number
        .PSECT LACODE,NOWRT
START:  ENTRY START,~m<>
        $ASSIGN_S DEVNAM=DEVNAME,CHAN=CHANNEL ; Assign channel
        BLBS RO,5$ ; No error
        BRW ERROR ; Error
5$: ; Set clock overflow rate
        ; To 2 khz. (1 mhz rate
        ; divided by 500 preset)
        $QIOW_S ,CHAN=CHANNEL,FUNC=#IO$_SETCLOCK,-
        IOB=IOB,,,P2=#1,P3=#^X143,P4#-500
        BLBC RO,ERROR ; Error
        MOVZWL IOB,RO ; Pick up I/O status
        BLBC RO,ERROR ; Error
        CLRW USW ; Start data transfer
        ; Clear USW (start with
        ; buffer 0)
        MOVL #100,COUNT ; Fill 100 buffers
        $QIOW_S ,CHANNEL,#IO$_STARTDATA,-
        IOB=IOB,,,P1=CBUFF,P2=#40,P3=#BFRAS
        BLBC RO,ERROR ; Error
; Note that the QIO waits until it finishes. Normally, the data is
; processed here as the buffers are filled. Check for error when
; the QIO completes.
        MOVZWL IOB,RO ; Pick up I/O status
        BLBC RO,ERROR ; Error
        RET ; All done - exit
ERROR: ; Enter here if error
        ; status in RO
        PUSHL RO ; Push onto stack
        CALLS #1,G^LIB$STOP ; Signal error
BFRAS: BFRAS,m^<> ; Buffer AST routine
        ; BFRAS is called whenever
        ; a buffer is filled
        .WORD 0
        INCB USW+1 ; Add 1 to buffer number
        CMPZV #0,#3,USW+1,#3 ; Handle wraparound
        BLEQ 10$
        CLRB USW+1 ; Use buffer 0
10$: DECL COUNT ; Decrement buffer count
        BGTR 20$
        BISB #^X40,USW+1 ; Enough buffers filled -
        ; Set stop bit
20$: BICB #^X80,USW+1 ; Clear done bit
        RET
        .END START

```

; *****

5 Line Printer Driver

This chapter describes the use of the line printer drivers LPDRIVER and LCDRIVER.

5.1 Supported Line Printer Devices

The following sections describe the line printer controllers and line printers supported by the VMS operating system.

5.1.1 LP11 Line Printer Controller

The LP11 line printer controller provides an interface between the VAX UNIBUS adapter and the line printer. The LP11 performs the following functions:

- Synchronizes single-character data transfers from the UNIBUS to the printer
- Informs the VMS operating system about printer status
- Enables the printer to gain control of the UNIBUS to report interrupts

5.1.2 DMF32 and DMB32 Line Printer Controllers

The DMF32 and DMB32 line printer controllers provide a direct memory access (DMA) interface between the VAX UNIBUS adapter (for the DMF32), or the VAXBI adapter (for the DMB32), and the line printer. The DMF32/DMB32 optionally perform the following functions:

- Tab expansion
- Carriage control
- Line wrapping and truncation
- Case conversion
- Passall mode
- Printall mode

5.1.3 LP27 Line Printer

The LP27 line printer is a high-speed, 132-column line printer, available with either a 64- or 96-character ASCII print set. The LP27-U is a fully buffered model that operates at a standard speed of up to 1200 lines per minute. Forms with up to six parts can be used for multiple copies. A version of the LP27 is available for operation of the printer up to 24.5 meters (1000 feet) from the host.

Line Printer Driver

5.1 Supported Line Printer Devices

5.1.4 LA11 DECprinter I

The LA11 DECprinter I is a medium-speed printer that operates at a standard speed of 180 characters per second. It provides a forms length switch to set the top of form to any of 11 common lengths, a paper-out switch and alarm, and a variable forms width. The LA11 uses a 96-character ASCII set; the column width is 132 characters.

5.1.5 LN01 Laser Page Printer

The LN01 laser page printer is a nonimpact printer that employs laser technology to produce high-quality print. Using electrophotographic imaging and xerographic printing, the LN01 prints one page at a time at a rate of 12 pages per minute. The print resolution of 300 x 300 dots per square inch produces characters of even density and alignment. The LN01 uses two, 188-character, fixed-space fonts; the column width is 132 characters.

5.1.6 LN03 Laser Page Printer

The LN03 laser page printer is a table-top, nonimpact page printer that uses laser imaging and xerographic printing techniques. The LN03 has a printing speed of 8 pages per minute with a print resolution of 300 x 300 dots per square inch. Four built-in fonts are available. Several column widths, including 80 or 132 characters, are also available.

5.2 Driver Features

The line printer drivers provide output character formatting and error recovery. These features are described in the following sections.

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):

- Rubouts are discarded.
- Tabs move the horizontal print position to the next MODULO (8) position unless the LP\$M_TAB characteristic is clear.
- All lowercase alphabetic characters are converted to uppercase before printing (unless the characteristic specifying lowercase characters is set; see Section 5.4.3 and Table 5-2).
- On printers where the line-feed, form-feed, vertical-tab, and carriage-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. Carriage returns are always output on units that have the LP\$M_CR characteristic set (see Section 5.4.3 and Table 5-2).

Line Printer Driver

5.2 Driver Features

- The horizontal print position is incremented on the output of all characters, including the space character. Characters are discarded if the horizontal print position is equal to or greater than the carriage width, unless the LP\$M_WRAP characteristic is set or the LP\$M_TRUNCATE characteristic is clear (see Section 5.3).
- On printers without a mechanical form feed (the form-feed function characteristic is not set; see Section 5.4.3 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.
- 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 VMS line printer drivers perform 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 after 30 seconds. Successive messages, if they occur, are sent 1, 2, 4, 8, . . . minutes after the initial message.
- The current operation is retried every two seconds to test for a changed situation, such as 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.

5.3 Device Information

You can obtain information on printer characteristics by using the Get Device/Volume Information (\$GETDVI) system service. (See the *VMS System Services Reference Manual*.)

\$GETDVI returns line printer characteristics when you specify the item codes DVI\$_DEVCHAR and DVI\$_DEVDEPEND. Tables 5-1 and 5-2 list these characteristics. The \$DEVDEF macro defines the device-independent characteristics; the \$LPDEF macro defines the device-dependent characteristics. DVI\$_DEVDEPEND returns a longword field that contains the device-dependent characteristics in the three low-order bytes and the page length in the high-order byte. Maximum page length is 255.

DVI\$_DEVTYPE and DVI\$_DEVCLASS return the device type and class names, which are defined by the \$DCDEF macro. The device type is a value that corresponds to the printer, for example, LP\$_LP27 or LP\$_LA11. The device class for printers is DC\$_LP. DVI\$_DEVBUFSIZ returns the page width, which is a value in the range of 0 through 255 on a DMF32 controller and 0 through 65535 on an LP11 or a DMB32 controller.

Line Printer Driver

5.3 Device Information

Table 5–1 Printer Device-Independent Characteristics

Characteristic ¹	Meaning
Dynamic Bits (Conditionally Set)	
DEV\$_SPL	Device is spooled.
DEV\$_AVL	Printer is on line and available.
Static Bits (Always Set)	
DEV\$_REC	Device is record-oriented.
DEV\$_CCL	Carriage control is enabled.
DEV\$_ODV	Device is capable of output.

¹Defined by the \$DEVDEF macro.

Table 5–2 Device-Dependent Characteristics for Line Printers

Value ¹	Meaning
LP\$_CR	Printer requires carriage return. (See Section 5.2.1).
LP\$_FALLBACK	Printer translates multinational characters to a seven-bit equivalent representation if possible. Otherwise, an underscore character (_) replaces the character. LP\$_FALLBACK has no effect on physical block operations. See Appendix B for a list of multinational characters.
LP\$_LOWER	Printer can print lowercase characters. If this value is not set, all lowercase characters are converted to uppercase when output. (LP\$_LOWER has no effect on write physical block operations.)
LP\$_MECHFORM	Printer has mechanical form feed. This characteristic is used when variable form length is required, such as in 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\$_PASSALL	All output data is in binary (no data interpretation occurs). Data termination occurs when the buffer is full (default buffer size is 132 bytes). Character formatting is disabled.
LP\$_PRINTALL	All printing and nonprinting characters are transferred to the printer, while character formatting remains enabled.
LP\$_TAB	Printer enables tab expansion.
LP\$_TRUNCATE	Printer truncates records that are larger than the carriage width.
LP\$_WRAP	Printer wraps records that are larger than the carriage width. If a string of text is longer than the width specified in the second longword, the string is continued on the next line.

¹Defined by the \$LPDEF macro.

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 take 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- or 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
- 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. The default buffer size is 132 bytes.

If the printer is not set spooled, write virtual block and write logical block operations perform the same function. If the printer is set spooled, a write logical block function queues the I/O to the printer, and a write virtual block function queues the I/O to the intermediate device, usually a disk.

All lowercase characters are converted to uppercase if the characteristics of the selected printer 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 (LP\$_M_MECHFORM) characteristic. The number of line feeds generated depends on the current page position and the page length.

Section 5.2.1 describes character formatting in greater detail.

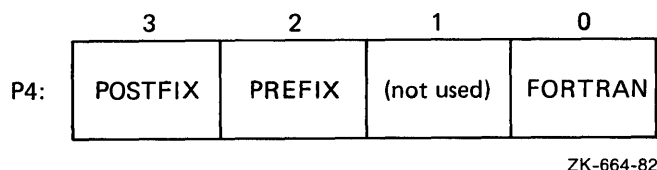
Line Printer Driver

5.4 Line Printer Function Codes

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 5-1 shows the P4 longword format.

Figure 5-1 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 as follows:

- 1 Prefix carriage control
- 2 Print
- 3 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 format.

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: carriage-return/line-feed combination ¹ , print buffer contents, return)
30	0	Double-space carriage control (Sequence: carriage-return/line-feed combination, carriage-return/line-feed combination, print buffer contents, return)
31	1	Page eject carriage control (Sequence: form feed, print buffer contents, return)

¹A carriage-return/line-feed combination is a carriage return followed by a line feed.

Line Printer Driver

5.4 Line Printer Function Codes

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

Byte 0 Value (hexadecimal)	ASCII Character	Meaning
2B	+	Overprint carriage control; allows double printing for emphasis or for special effects (Sequence: print buffer contents, return)
24	\$	Prompt carriage control (Sequence: carriage-return/line-feed combination, 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 carriage-return/line-feed combinations.	

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 (seven-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 (eight-bit character set; see Appendix B).

Figure 5-2 shows the prefix and postfix hexadecimal coding that produces the carriage control functions listed in Table 5-3. Prefix and postfix coding provides an alternative way to achieve these controls.

In the first example, the prefix/postfix hexadecimal coding for a single-space carriage control (carriage-return/line-feed combination, print buffer contents, carriage-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 Line Printer Function Codes

Figure 5-2 Write Function Carriage Control (Prefix and Postfix Coding)

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

ZK-665-82

5.4.2 Sense Printer Mode

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

Line Printer Driver

5.4 Line Printer Function Codes

5.4.3 Set Mode

Set mode operations affect the operation and characteristics of the associated line printer. The VMS operating system 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. The following function codes are provided:

- IO\$_SETMODE
- IO\$_SETCHAR

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

P1—The address of a characteristics buffer

Figure 5-3 shows the quadword P1 characteristics buffer for IO\$_SETMODE. Figure 5-4 shows the same buffer for IO\$_SETCHAR.

Figure 5-3 Set Mode Buffer

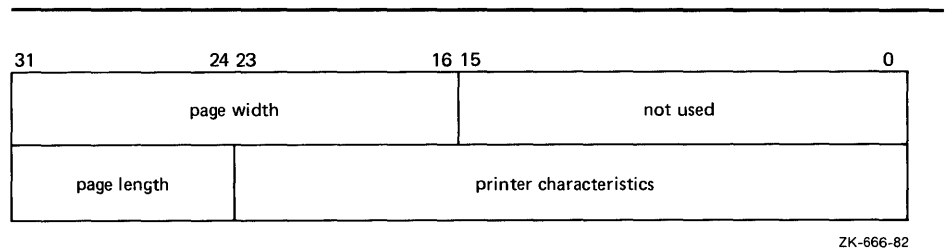
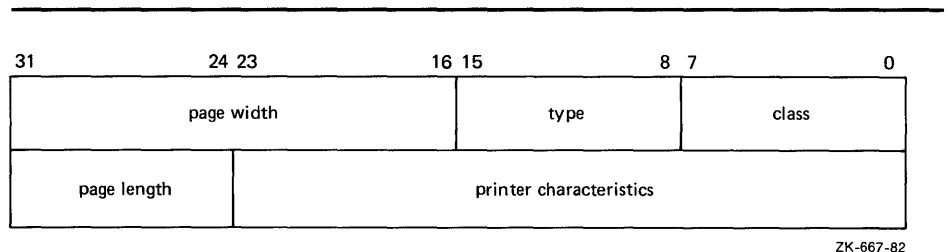


Figure 5-4 Set Characteristics Buffer



In the buffer, the device class is DC\$_LP. The printer type is a value that corresponds to the printer: DT\$_LP27 or DT\$_LA11. The type can be changed by the IO\$_SETCHAR function. The page width is a value in the range of 0 through 255 on a DMF32 controller and 0 through 65535 on an LP11 or DMB32 controller.

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

Line Printer Driver

5.4 Line Printer Function Codes

Application programs that change specific line printer characteristics should perform the following steps:

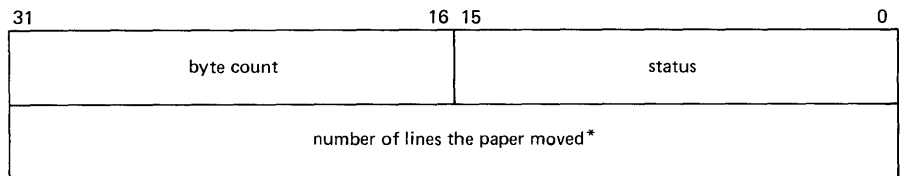
- 1 Use the `IO$_SENSEMODE` function to read the current characteristics.
- 2 Modify the characteristics.
- 3 Use the set mode function to write back the results.

Failure to follow this sequence will result in clearing any previously set characteristic.

5.5 I/O Status Block

The I/O status blocks (IOSB) for the write and set mode I/O functions are shown in Figures 5-5 and 5-6. Appendix A lists the status returns for these functions. (The *VMS System Messages and Recovery Procedures Reference Volume* provides explanations and suggested user actions for these returns.)

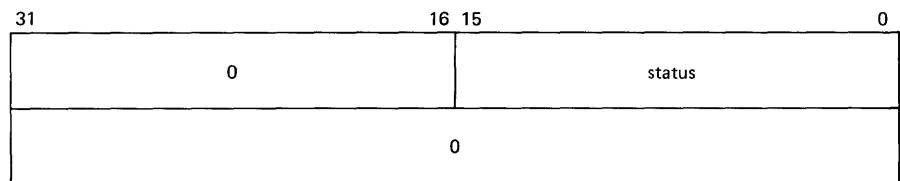
Figure 5-5 IOSB Contents — Write Function



*0 if `IO$_WRITEPBLK`

ZK-668-82

Figure 5-6 IOSB Contents — Set Mode Function



ZK-669-82

5.6 Programming Example

The following sample program (Example 5-1) 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`.

Line Printer Driver

5.6 Programming Example

Example 5-1 Line Printer Program Example

```
*****
;
;
; .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$: .ASCII /LINE_PRINTER/ ;Name string of output device
20$: ;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 set up 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
```

Example 5-1 Cont'd. on next page

Line Printer Driver

5.6 Programming Example

Example 5-1 (Cont.) Line Printer Program Example

```

;
; *****
;
;                               Start Program
;
; *****
;
; 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 ($QIOW) system service requests. 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 = 0, assign failure
;                               MOVL #11,R3 ;Set up loop count
;                               MOVAL OUTPUT_FORMAT,R2 ;Set up o/p format address
;                               ;in R2
;
; Start the printing loop.
;
30$: $QIOW_S CHAN=DEVICE_CHANNEL,- ;Print on device channel
;                               FUNC=#IO$_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 autoincrement
;                               BLBC R0,40$ ;If low bit = 0, I/O failure
;                               SOBGTR R3,30$ ;Branch if not finished
40$: $DASSGN_S CHAN=DEVICE_CHANNEL ;Deassign channel
50$: RET ;Return
;
;                               .END PRINTER_EXAMPLE

```

6 Magnetic Tape Drivers

This chapter describes the use of the VMS magnetic tape drivers. These drivers support the devices listed in Table 6-1.

Table 6-1 Supported Magnetic Tape Devices

Controller	Drive ¹	Code	No. of Tracks	Recording Density (bpi)	Tape Speed (ips)	Max. Data Transfer Rate (bytes per second)	Recording Method ²
TS11	TS04	MS	9	1600	45	72,000	PE
TM03	TE16	MT	9	800 or 1600	45	36,000 (for 800 bpi); 72,000 (for 1600 bpi)	NRZI or PE
	TU45	MT	9	800 or 1600	75	60,000 (for 800 bpi); 120,000 (for 1600 bpi)	NRZI or PE
	TU77	MT	9	800 or 1600	125	100,000 (for 800 bpi); 200,000 (for 1600 bpi)	NRZI or PE
TM78	TU78	MF	9	1600 or 6250	125	200,000 (for 1600 bpi); 781,250 (for 6250 bpi)	PE or GCR
³	TU80	MS	9	1600	25 or 100	160,000	PE
³	TU81 TU81-Plus TQ81	MU	9	1600 or 6250	25 or 75	120,000 (for 1600 bpi); 468,750 (for 6250 bpi)	PE or GCR
HSC50	TA81	MU	9	1600 or 6250	25 or 75	120,000 (for 1600 bpi); 468,750 (for 6250 bpi)	PE or GCR
	TA78	MF	9	1600 or 6250	125	200,000 (for 1600 bpi); 781,250 (for 6250 bpi)	PE or GCR
	TA79	MU	9	1600 or 6250	125	769,000	PE or GCR
TUK50 TQK50 TZK50	TK50 ⁵	MU	22 ⁴	6666	75	45,000	MFM
LESI Adapter	RV20	MU	9	6250	N/A	1.33 Megabytes	Write once optical disk

¹The TU81, TU81-Plus, TA78, TU78, TA81, and TF30 are tape mass storage control protocol (TMSCP) drives. The TZ30 is an SCSI drive.

²NRZI = non-return-to-zero-inverted; PE = phase encoded; GCR = group-coded recording; MFM = modified frequency modulation; HDMFM = high density modified frequency modulation

³Has a self-contained controller

⁴Each track written separately—not in parallel

⁵The TK50 is a tape mass storage control protocol (TMSCP) device when configured on (. . . BA23 BA123 w/s) systems. The TK50 has a self-contained controller when configured on VAXstation 2000 and MicroVAX 2000 systems.

Magnetic Tape Drivers

Table 6–1 (Cont.) Supported Magnetic Tape Devices

Controller	Drive ¹	Code	No. of Tracks	Recording Density (bpi)	Tape Speed (ips)	Max. Data Transfer Rate (bytes per second)	Recording Method ²
³	TF30 TZ30	MU	22 ⁴	6666	75	45,000	MFM
TQK70	TK70	MU	48	10000	100	90,000	HDMFM
TM79	TU79	MF	9	1600 or 6250	125	769,000	PE or GCR

¹The TU81, TU81-Plus, TA78, TU78, TA81, and TF30 are tape mass storage control protocol (TMSCP) drives. The TZ30 is an SCSI drive.

²NRZI = non-return-to-zero-inverted; PE = phase encoded; GCR = group-coded recording; MFM = modified frequency modulation; HDMFM = high density modified frequency modulation

³Has a self-contained controller

⁴Each track written separately—not in parallel

6.1 Supported Magnetic Tape Controllers

The following sections describe the VMS magnetic tape controllers.

6.1.1 TM03 Magnetic Tape Controller

The TM03 magnetic tape controller supports up to eight TE16, TU45, or TU77 tape drives. These dual-density (800 or 1600 bpi) drives differ in speed: the TE16, TU45, and TU77 read and write data at 45, 75, and 125 inches per second, respectively. Each drive can hold one 2400-foot, 9-track reel with a capacity of approximately 40 million characters. The TM03 controller is connected to the MASSBUS through a MASSBUS adapter.

6.1.2 TS11 Magnetic Tape Controller

The TS11 magnetic tape controller connects to the UNIBUS through a UNIBUS adapter and supports one TS04 tape drive. The TS11/TS04 is a single-density tape system that supports 1600-bpi, phase-encoded recording.

6.1.3 TM78 Magnetic Tape Controller

The TM78 magnetic tape controller supports up to four TU78 tape drives. These high-performance, dual-density drives (1600 or 6250 bpi) operate at 125 inches per second (ips) using a 2400-foot reel of tape with a capacity of approximately 146 million characters when recorded in the GCR (6250 bpi) mode. The TM78 controller is connected to the MASSBUS through a MASSBUS adapter.

Magnetic Tape Drivers

6.1 Supported Magnetic Tape Controllers

6.1.4 TU80 Magnetic Tape Subsystem

The TU80 is a single-density, dual-speed (25 or 100 ips) magnetic tape subsystem that uses streaming tape technology (see Section 6.2.4). It supports one drive per subsystem. The TU80 connects to the UNIBUS through a UNIBUS adapter and completely emulates the TS11 magnetic tape controller.

6.1.5 TU81 and TA81 Magnetic Tape Subsystems

The TU81 and the TA81 are high-performance, dual-density (1600 or 6250 bpi), dual-speed (25 or 75 ips) magnetic tape subsystems that use streaming tape technology (see Section 6.2.4). The TU81 connects to the UNIBUS through a UNIBUS adapter. The TA81 attaches to an HSC50 controller. Both drives are managed with the tape mass storage control protocol (TMSCP).

6.1.6 TK50 Cartridge Tape System

The TK50 is a 5.24-inch, 95-megabyte cartridge tape that uses streaming tape technology (see Section 6.2.4). The TK50 records data serially on 22 tracks using serpentine recording, rather than on separate (parallel) tracks. Data written to tape is automatically read as it is written. A CRC check is performed and the controller is notified immediately if an error occurs on the tape. The TQK50 is a dual-height Q-BUS controller for the TK50 tape drive. The TUK50 is a UNIBUS controller for the same drive. Both the TQK50 and the TUK50 are TMSCP devices. The TZK50 is not a TMSCP device; it contains its own internal controller. Only one drive is supported per controller.

6.2 Driver Features

The 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, an RP05 disk and a TM03 tape formatter
- Reverse read function (except for the TK50 on TUK50 and TQK50 controllers)
- Reverse data check function (except for TS11, and TK50 on TUK50 and TQK50 controllers)
- Data checks on a per-request, per-file, or per-volume basis (except for TS11)
- Full recovery from power failure for online drives with volumes mounted, including repositioning by the driver (except on VAXstation 2000 and MicroVAX 2000 systems)
- 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
- Online diagnostic support for drive level diagnostics

Magnetic Tape Drivers

6.2 Driver Features

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

6.2.1 Master Adapters and Slave Formatters

The VMS operating system supports the use of many 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.

The VMS operating system also supports the use of multiple slave formatters per master adapter on a system. For example, more than one TM03 or TM78 magnetic tape formatter per MBA can be used on a system. A slave formatter accepts data and commands from a master adapter and directs the operation of one or more slave drives. The TM03 and the TM78 are slave formatters. The TE16, TU45, TU77, and TU78 magnetic tape drives are slave drives.

6.2.2 Data Check

After successful completion of an I/O operation, a data check is made to compare the data in memory with that on the tape. After a write or read (forward) operation, the tape drive spaces backward, and then performs a write check data operation. After a read operation in the reverse direction, the tape drive spaces forward, and then performs a write check data reverse operation. With the exception of TS04 and TU80 drives, magnetic tape drivers support data checks at the following three levels:

- Per request—You 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—You can specify the characteristics “data check all reads” and “data check all writes” when the volume is mounted. The *VMS DCL Dictionary* describes volume mounting and dismounting. The *VMS System Services Reference Manual* describes the Mount Volume (\$MOUNT) and Dismount Volume (\$DISMOU) system services.
- Per file—You 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 1 of this manual and the *VMS Record Management Services Manual* both describe file access.

Data check is distinguished from a BACKUP/VERIFY operation, which writes an entire save set, rewinds, and then compares the tape to the original tape.

See Section 6.1.6 for information on TK50 data check.

Note: Read and write operations with data check can result in very slow performance on streaming tape drives.

6.2.3 Error Recovery

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

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

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

Power failure recovery consists of repositioning the reel to the position held at the start of the I/O operation in progress at the time of the power failure, and then reexecuting this operation. This repositioning might or might not require operator intervention to reload the drives. When such operator intervention is required, “device not ready” messages are sent to the operator console to solicit reloading of mounted drives. Power failure recovery is not supported on VAXstation 2000 and MicroVAX 2000 systems.

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.

If a nonfatal controller/drive error occurs, the driver (or the controller, depending on the type of drive) attempts to reexecute the I/O operation up to 16 times before returning a fatal error. The driver repositions the tape before each retry.

The inhibit retry function modifier (IO\$M_INHRETRY) inhibits all normal (nonspecial conditions) error recovery. If an error occurs, and the request includes that modifier, the operation is immediately terminated and the driver returns a failure status. IO\$M_INHRETRY has no effect on power failure and timeout recovery.

The driver can write up to 16 extended interrecord gaps during the error recovery for a write operation. For the TE16, TU45, and TU77, writing these gaps can be suppressed by specifying the inhibit extended interrecord gap function modifier (IO\$M_INHEXTGAP). This modifier is ignored for the other magnetic tape drives.

6.2.4 Streaming Tape Systems

Streaming tape systems (TU80, TU81, TU81-Plus, TA81, TK50, TK70, TF30, and TZ30) use the supply and takeup reel mechanisms to control tape speed and tension directly, thereby eliminating the need for more complex and costly tension and drive components. Streaming tapes have a very simple tape path, much like a home audio reel-to-reel recorder.

Note: Read and write operations with data check can result in very slow performance on streaming tape drives.

Because the motors driving the reels are low-powered, and because there is no tape buffering, streaming tape drives are not capable of starting and stopping in the interrecord gaps like conventional tape drives. When a streaming tape does have to stop, the following events occur:

- 1 The tape slowly coasts forward to a stop.

Magnetic Tape Drivers

6.2 Driver Features

- 2 It backs up over a section previously processed.
- 3 It halts to await the next command.
- 4 It accelerates so that, when the original interrecord gap is encountered, the tape is moving at full speed.

These steps, allowing the tape to reposition, require approximately one-half second to complete on TU8x tapes and about three seconds on TK50 tapes. If the operating system is not capable of writing to, or reading from, a streaming tape drive at a rate that will keep the drive in constant motion (streaming) the drive repositions itself when it runs out of commands to execute. That produces a situation known as *thrashing*, in which the relatively long reposition times exceed the time spent processing data and the result is lower-than-expected data throughput.

Thrashing is entirely dependent on how fast the system can process data relative to the tape drive speed while streaming. Consequently, the greatest efficiency is obtained when you provide sufficient buffering to ensure continuous tape motion. Some streaming tape drives supported by the VMS operating system (TU80, TU81, TU81-Plus, and TA81) are dual-speed devices that automatically adjust the tape speed to maximize data throughput and minimize thrashing.

The TK50 writes up to seven filler records to keep the tape in motion. These records are ignored when the data is read.

6.3 Device Information

You can obtain information on all magnetic tape device characteristics by using the Get Device/Volume Information (\$GETDVI) system service. (See the *VMS System Services Reference Manual*.)

\$GETDVI returns magnetic tape characteristics when you specify the item codes DVI\$_DEVCHAR, DVI\$_DEVCHAR2, DVI\$_DEVDEPEND, and DVI\$_DEVDEPEND2. Tables 6-2, 6-3, and 6-4 list these characteristics. The \$DEVDEF macro defines the device-independent characteristics, the \$MTDEF macro defines the device-dependent characteristics, and the \$MT2DEF macro defines the extended device characteristics. The extended device characteristics apply only to the TU81-Plus.

Table 6-2 Magnetic Tape Device-Independent Characteristics

Characteristic ¹	Meaning
Dynamic Bits (Conditionally Set)	
DEV\$_AVL	Device is online and available.
DEV\$_FOR	Volume is foreign.
DEV\$_MNT	Volume is mounted.
DEV\$_RCK	Perform data check on all read operations.
DEV\$_WCK	Perform data check on all write operations.

¹Defined by the \$DEVDEF macro.

Magnetic Tape Drivers

6.3 Device Information

Table 6–2 (Cont.) Magnetic Tape Device-Independent Characteristics

Characteristic ¹	Meaning
Static Bits (Always Set)	
DEV\$_M_FOD	Device is file-oriented.
DEV\$_M_IDV	Device is capable of input.
DEV\$_M_ODV	Device is capable of output.
DEV\$_M_SQD	Device is capable of sequential access.
DEV\$_M_WBC ²	Device is capable of write-back caching.

¹Defined by the \$DEVDEF macro.

²This bit is located in DVI\$_DEVCHAR2

Table 6–3 Device-Dependent Information for Tape Devices

Characteristic ¹	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 the tape in the forward direction.
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 non-return-to-zero-inverted 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 as follows: MT\$_K_GCR_6250 Group-coded recording, 6250 bpi MT\$_K_PE_1600 Phase-encoded recording, 1600 bpi MT\$_K_NRZI_800 Non-return-to-zero-inverted recording, 800 bpi MT\$_K_BLK_833 Cartridge block mode recording ²
MT\$_V_FORMAT MT\$_S_FORMAT	Specifies the format in which all data transfers are performed. A possible format value is as follows: MT\$_K_NORMAL11 Normal PDP–11 format. Data bytes are recorded sequentially on tape with each byte occupying exactly one frame.

¹Defined by the \$MTDEF macro.

²Only for the TK50, TF30, and TZ30

Magnetic Tape Drivers

6.3 Device Information

Table 6–4 Extended Device Characteristics for Tape Devices

Characteristic ¹	Meaning
MT2\$_V_WBC_ENABLE	If set, write-back caching is enabled for this unit.
MT2\$_V_RDC_DISABLE	If set, read caching is disabled for this unit.

¹Defined by the \$MT2DEF macro. Only for the TU81-Plus. Initial device status will show both of these bits cleared; write-back caching will be disabled, read caching will be enabled.

DVI\$_DEVTYPE and DVI\$_DEVCLASS return the device type and class names, which are defined by the \$DCDEF macro. DVI\$_DEVBUFSIZ returns the buffer size. The buffer size is the default to be used for tape transfers (normally 2048 bytes). The device class for magnetic tapes is \$DCTAPE, and the device type is determined by the magnetic tape model. For example, the device type for the TA78 is DT\$_TA78, for the TA81 it is DT\$_TA81.

6.4 Magnetic Tape Function Codes

The VMS magnetic tape driver can perform logical, virtual, and physical I/O functions. Foreign-mounted devices do not require privilege to perform logical and virtual I/O requests.

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. The results of logical and physical I/O operations are unpredictable if an ACP is present.

Virtual I/O functions require intervention by an ACP and must be executed in a prescribed order. The normal order is to create and access a file, write information to that file, and deaccess the file. Subsequently, when you access the file, you read the information and then deaccess the file. You can write over the file when the information it contains is no longer useful and the file has expired.

Any number of bytes (from a minimum of 14 to a maximum of 65,535) 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.

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

Table 6–5 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. Chapter 1 describes the QIO level interface to the magnetic tape device ACP.

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

Table 6–5 Magnetic Tape I/O Functions

Function Code and Arguments	Type ¹	Function Modifiers	Function
IO\$_CREATE P1,- [P2],[P3],[P4],- [P5]	V	IO\$_CREATE IO\$_ACCESS	Create a file.
IO\$_ACCESS P1,- P2],[P3],[P4],- [P5]	V	IO\$_CREATE IO\$_ACCESS	Search a tape for a specified file and access the file if found and IO\$_ACCESS is set. If the file is not found and IO\$_CREATE is set, create a file at end-of-tape (EOT) marker.
IO\$_DEACCESS P1,- [P2],[P3],[P4],- [P5]	V		Deaccess a file and, if the file has been written, write out trailer records.
IO\$_DSE ²	P	IO\$_NOWAIT	Erase a prescribed section of the tape.
IO\$_MODIFY P1,- [P2],[P3],[P4],- [P5]	V		Write user labels.
IO\$_READVBLK P1,P2	V	IO\$_DATACHECK ³ IO\$_INHRETRY IO\$_REVERSE ⁴	Read virtual block.
IO\$_READLBLK P1,P2	L	IO\$_DATACHECK ³ IO\$_INHRETRY IO\$_REVERSE ⁴	Read logical block.
IO\$_READPBLK P1,P2	P	IO\$_DATACHECK ³ IO\$_INHRETRY IO\$_REVERSE ⁴	Read physical block.
IO\$_WRITEVBLK P1,P2	V	IO\$_DATACHECK ³ IO\$_INHRETRY IO\$_INHEXTGAP ⁷ IO\$_NOWAIT ⁵	Write virtual block.
IO\$_WRITELBK P1,P2	L	IO\$_ERASE ⁶ IO\$_DATACHECK ³ IO\$_INHRETRY IO\$_INHEXTGAP ⁷ IO\$_NOWAIT ⁵	Write logical block.
IO\$_WRITEPBLK P1,P2	P	IO\$_ERASE ⁶ IO\$_DATACHECK ³ IO\$_INHRETRY IO\$_INHEXTGAP ⁷ IO\$_NOWAIT ⁵	Write physical block.

¹V = virtual; L = logical; P = physical

²Only for TMSCP drives, TZK50, and TZ30

³Not for TS04 and TU80

⁴Not for TUK50 and TQK50

⁵Only for TU81-Plus drives

⁶Takes no arguments; valid only for TMSCP drives, TZK50, and TZ30

⁷Only for TE16, TU45, and TU77

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

Table 6–5 (Cont.) Magnetic Tape I/O Functions

Function Code and Arguments	Type ¹	Function Modifiers	Function
IO\$_REWIND	L	IO\$_INHRETRY IO\$_NOWAIT	Reposition tape to the beginning-of-tape (BOT) marker.
IO\$_REWINDOFF	L	IO\$_INHRETRY IO\$_NOWAIT	Rewind and unload the tape on the selected drive.
IO\$_UNLOAD	L	IO\$_INHRETRY IO\$_NOWAIT	Rewind and unload the tape on the selected drive.
IO\$_SKIPFILE P1	L	IO\$_INHRETRY IO\$_NOWAIT ⁵	Skip past a specified number of tape marks in either a forward or reverse direction.
IO\$_SKIPRECORD P1	L	IO\$_INHRETRY IO\$_NOWAIT ⁵	Skip past a specified number of blocks in either a forward or reverse direction.
IO\$_WRITEOF	L	IO\$_INHRETRY IO\$_INHEXTGAP ⁷ IO\$_NOWAIT ⁵	Write an extended interrecord gap followed by a tape mark.
IO\$_PACKACK	P		Initialize volume valid bit.
IO\$_AVAILABLE	P		Clear volume valid bit.
IO\$_SENSEMODE [P1]- [P2] ⁸	L	IO\$_INHRETRY	Sense the tape characteristics and return them in the I/O status block.
IO\$_SENSECHAR [P1]- [P2] ⁸	P	IO\$_INHRETRY	Sense the tape characteristics and return them in the I/O status block.
IO\$_SETMODE P1,- [P2] ⁸	L		Set tape characteristics for subsequent operations.
IO\$_SETCHAR P1,- [P2] ⁸	P		Set tape characteristics for subsequent operations.
IO\$_ACPCONTROL P1,- [P2],[P3],[P4],- [P5]	V	IO\$_DMOUNT	Perform miscellaneous control functions. ⁹

¹V = virtual; L = logical; P = physical

⁵Only for TU81-Plus drives

⁷Only for TE16, TU45, and TU77

⁸The P1 and P2 arguments for IO\$_SENSEMODE and IO\$_SENSECHAR and the P2 argument for IO\$_SETMODE and IO\$_SETCHAR are for TMSCP drives only

⁹See Section 1.6.7 for additional information.

The function-dependent arguments for IO\$_CREATE, IO\$_ACCESS, IO\$_DEACCESS, IO\$_MODIFY, IO\$_ACPCONTROL are as follows:

- P1—The address of the file information block (FIB) descriptor.
- P2—Optional. The address of the file name string descriptor. 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—Optional. The address of the word that is to receive the length of the resultant file name string.

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

- P4—Optional. The address of a descriptor for a buffer that is to receive the resultant file name string.
- P5—Optional. The address of a list of attribute descriptors. 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.

See Chapter 1 for more information on these functions.

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

- 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 length of the buffer specified by P1

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.)

The following example shows the correct method of defining the P1 parameter in a IO\$_SKIPRECORD QIO.

```

.
.
.
TAPE_CHAN:
    .WORD    0
IOSB:     .WORD    0
          .WORD    0
          .LONG    0
DEVICE:   .ASCID   /$127$MUAO:/
RECORD:   .LONG    2000
;
          .PSECT   CODE, EXE, NOWRT
;
          .ENTRY   MT_IO, ^M<>
;
$ASSIGN_S  CHAN=TAPE_CHAN, -
           DEVNAM=DEVICE
BLBC       RO, EXIT_ERROR
;
$QIOW_S    CHAN=TAPE_CHAN, -
           FUNC=#IO$_SKIPRECORD, -
           IOSB=IOSB, -
           P1=RECORD
BLBC       RO, EXIT_ERROR
$EXIT_S    RO
.
.
.
EXIT_ERROR:
    $EXIT_S    RO
    .END      MT_IO

```

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

6.4.1 Read

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

The VMS operating system provides the following 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 directed to a volume that is mounted foreign, it is converted to a read logical block function. If a read virtual block function is directed to a volume that is mounted structured, the volume is handled the same way as a file-structured device.

Two function-dependent arguments are used with these codes: P1 and P2. These arguments are described in Section 6.4.

If the read function code includes the reverse function modifier (IO\$_M_REVERSE), the drive reads the tape in the reverse direction instead of the forward direction. IO\$_M_REVERSE cannot be specified for the TUK50 and TQK50 devices.

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 operation completes. (The drive performs a space reverse or space forward between the read and data check operations.) A data check operation is also performed if the volume that was 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 TS04 and TU80 tape drives do not support the data check function.

For read physical block and read logical block functions, the drive returns the status SS\$_NORMAL (not end-of-tape status) if either of the following conditions occurs and no other error condition exists:

- The tape is positioned past the end-of-tape (EOT) position at the start of the read (forward or reverse) operation.
- The tape enters the EOT region as a result of the read (forward) operation.

The transferred byte count reflects the actual number of bytes read.

If the drive reads a tape mark during a logical or physical read operation in either the forward or reverse direction, any of the following conditions can return an end-of-file status:

- The tape is positioned past the EOT position at the start of the read operation.
- The tape enters the EOT region as a result of the read operation.
- The drive reads a tape mark as a result of a read operation but the tape does not enter the EOT region.

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

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

If the drive attempts to read a block that is larger than the specified memory buffer during a logical or physical read operation, a data overrun status is returned. The buffer receives only the first part of the block. On a read in the reverse direction (on drives other than the TK50 and TZ30) the buffer receives only the latter part of the block. 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. The TUK50 and TZ30 cannot actually perform read reverse operations; they must be simulated by the driver. Therefore, the data returned are those that would have been returned had the block been read in the forward direction.

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

6.4.2 Write

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

The VMS operating system provides the following 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 directed to a volume that is mounted foreign, the function is converted to a write logical block. If a write virtual block function is directed to a volume that is mounted structured, the volume is handled the same way as a file-structured device.

Two function-dependent arguments are used with these codes: P1 and P2. These arguments are described in Section 6.4.

The IO\$_M_ERASE function modifier can be used with the IO\$_WRITELBLK and IO\$_WRITEPBLK function codes to erase a user-selected part of a tape. This modifier propagates an erase pattern of all zeros from the current tape position to 10 feet past the EOT position and then rewinds to the BOT marker.

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 operation completes. (The drive performs a space reverse between the write and the data check operations.) The driver forces a data check operation when an error occurs during a write operation. This ensures that the data can be reread. 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 TS04 and TU80 tape drives do not support the data check function.

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

If the IO\$M_NOWAIT function modifier is specified, write-back caching is enabled on a per command basis. IO\$M_NOWAIT is applicable only to TU81-Plus drives.

If the drive performs a write physical block or a write logical block operation, an EOT status is returned if either of the following conditions occurs and no other error condition exists:

- The tape is positioned past the EOT position at the start of the write operation.
- The tape enters the EOT 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.

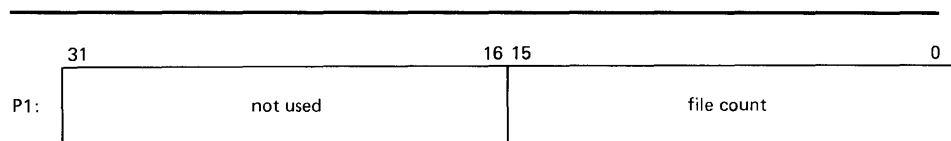
6.4.3 Rewind

The rewind function repositions the tape to the beginning-of-tape (BOT) marker. If the IO\$M_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.

6.4.4 Skip File

The skip file 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 6-1. 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 as a signed number in the I/O status block.)

Figure 6-1 IO\$_SKIPFILE Argument



ZK-671-82

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. A negative skip file function leaves the tape positioned just before a tape mark (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.

A skip file function in the forward direction can also be terminated if two consecutive tape marks are encountered. Section 6.4.5.1 describes this feature.

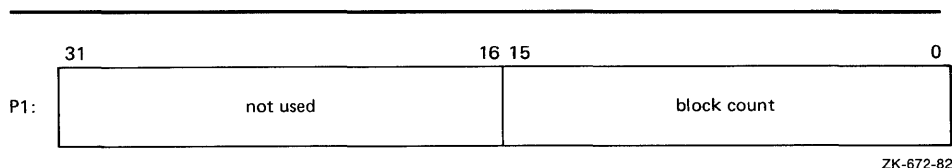
Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

6.4.5 Skip Record

The skip record function skips past a specified number of physical tape blocks in either a forward or reverse direction. A device- or function-dependent argument (P1) specifies the number of blocks to skip, as shown in Figure 6-2. 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 as a signed number in the I/O status block. If a tape mark is detected, the count is the number of blocks skipped, plus 1 (forward tape motion) or minus 1 (reverse tape motion).

Figure 6-2 IO\$_SKIPRECORD Argument



A skip record operation is terminated by the end-of-file marker 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.

A skip record function in the forward direction can also be terminated if the tape was originally positioned between two tape marks. Section 6.4.5.1 describes this feature.

6.4.5.1 Logical End-of-Volume Detection

A skip file or skip record operation is terminated when both of the following conditions exist:

- The tape is mounted foreign.
- Two consecutive tape marks are encountered when the tape moves in the forward direction.

After the operation terminates, the tape remains positioned between the two tape marks that were detected. The I/O status block (IOSB) returns the status SS\$_ENDOFVOLUME and the actual number of files (or records) skipped during the operation prior to the detection of the second tape mark. The skip count is returned in the high-order word of the first longword of the IOSB.

Subsequent skip record (or skip file) requests terminate immediately when the tape is positioned between the two tape marks, producing no net tape movement and returning the SS\$_ENDOFVOLUME status with a skip count of zero.

To move the tape beyond the second tape mark, you must employ another I/O function. For example, the IO\$_READLBLK function, if issued after receipt of the SS\$_ENDOFVOLUME status return, terminates with an SS\$_ENDOFFILE status and with the tape positioned just past the second tape mark. From this new position, other skip functions could be issued to produce forward tape motion (assuming there is additional data on the tape).

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

If three consecutive tape marks are encountered during a skip file function, you must issue two IO\$_READLBLK functions, the first to get the SS\$_ENDOFFILE return, the second to position the tape past the third tape mark.

6.4.6 Write End-of-File

The write-end-of-file function writes an extended interrecord gap (of approximately 3 inches for non-return-to-zero-inverted (NRZI) recording and 1.5 inches for phase-encoded (PE) recording) followed by a tape mark. No device- or function-dependent arguments are used with IO\$_WRITEOF.

An end-of-tape (EOT) 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 EOT region.

6.4.7 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 operation is initiated. No device- or function-dependent arguments are used with IO\$_REWINDOFF.

6.4.8 Unload

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

6.4.9 Sense Tape Mode

The sense tape mode function senses the current device-dependent and extended device characteristics (see Tables 6-3 and 6-4).

The VMS operating system provides the following function codes:

- IO\$_SENSEMODE—Sense mode
- IO\$_SENSECHAR—Sense characteristics

Sense mode requires logical I/O privilege. Sense characteristics requires physical I/O privilege. For TMSCP drives the sense mode function returns magnetic tape information in a user-supplied buffer, which is specified by the following function-dependent arguments:

- P1—Optional. Address of a user-supplied buffer.
- P2—Optional. Length of user-supplied buffer.

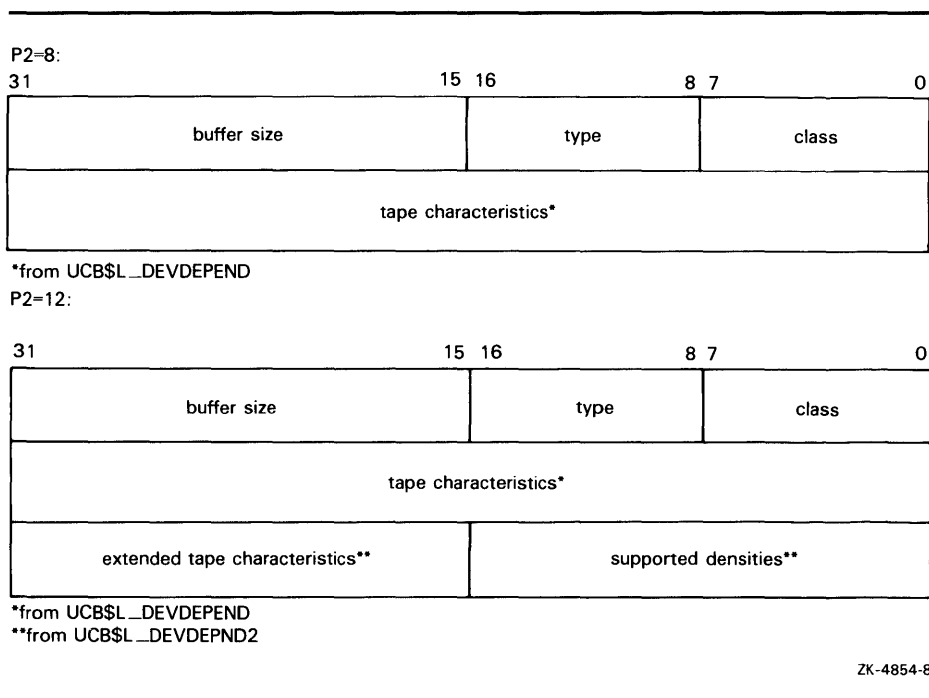
Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

If P1 is not zero, the sense mode buffer returns the tape characteristics. (If P2=8, the second longword of the buffer contains the device-dependent characteristics. If P2=12, the second longword contains the device-dependent characteristics and the third longword contains the tape densities that the drive supports and the extended tape characteristics.) The extended characteristics are identical to the information returned by DVI\$_DEVDEPEND2 (see Table 6-4). Figure 6-3 shows the contents of the P1 buffer.

Regardless of whether the P1 buffer is specified, the I/O status block returns the device-dependent characteristics in the second longword (see Figure 6-6). These characteristics are identical to the information returned by DVI\$_DEVDEPEND (see Table 6-3 in Section 6.3).

Figure 6-3 Sense Mode P1 Buffer



6.4.10 Set Mode

Set mode operations affect the operation and characteristics of the associated magnetic tape device. The VMS operating system defines 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. The following function codes are provided:

- IO\$_SETMODE—Set mode
- IO\$_SETCHAR—Set characteristics

These functions take the following device- or function-dependent arguments (other arguments are ignored):

- P1—The address of a characteristics buffer

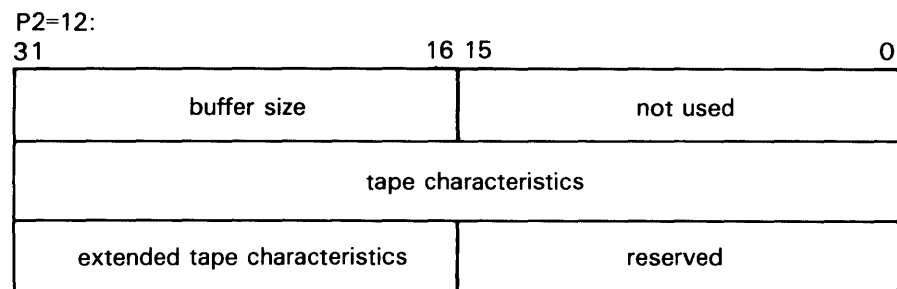
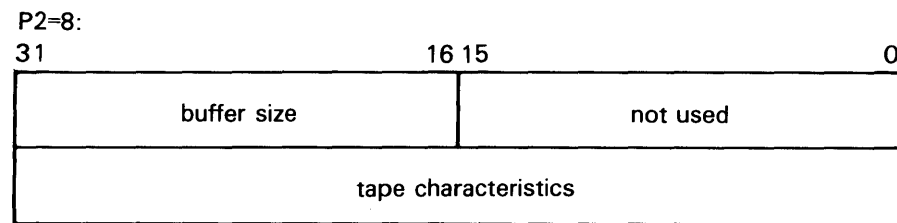
Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

- P2—Optional. The length of the characteristics buffer. Default is eight bytes. If a length of 12 bytes is specified, the third longword (which is for TMSCP drives only) specifies the extended tape characteristics.

Figure 6-4 shows the P1 characteristics buffer for IO\$_SETMODE. Figure 6-5 shows the same buffer for IO\$_SETCHAR.

Figure 6-4 Set Mode Characteristics Buffer

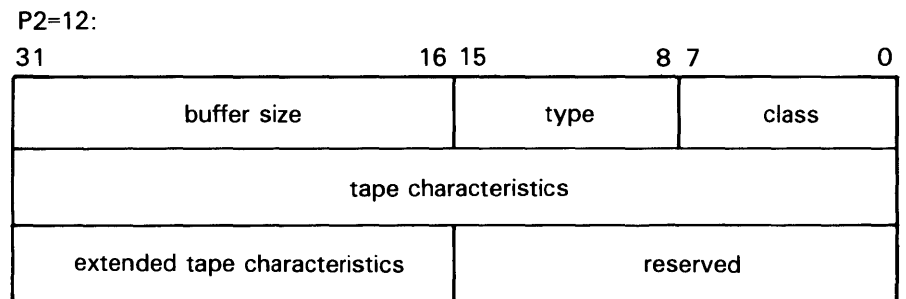
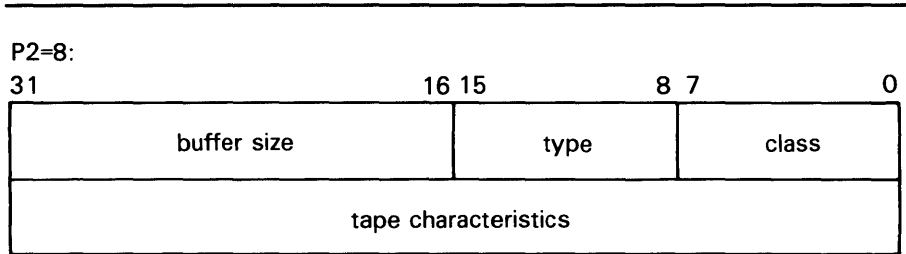


ZK-4856-85

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

Figure 6–5 Set Characteristics Buffer



ZK-4855-85

The first longword of the P1 buffer for the set characteristics function contains information on device class and type, and the buffer size. The device class for tapes is DC\$_TAPE.

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 second longword of the P1 buffer for both the set mode and set characteristics functions contains the tape characteristics. Table 6–6 lists the tape characteristics and their meanings. The \$MTDEF macro defines the symbols listed. If P2=12, the third longword contains the extended tape characteristics for TMSCP drives, which are listed in Table 6–7. The extended tape characteristics are defined by the \$MT2DEF macro and are identical to the information returned by DVI\$_DEVDEPEND2.

Table 6–6 Set Mode and Set Characteristics Magnetic Tape Characteristics

Characteristic ¹	Meaning
MT\$_PARITY	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 non-return-to-zero-inverted recording at 800 bpi. Even parity cannot be selected for phase-encoded recording (tape density is MT\$_PE_1600) or group-coded recording (tape density is MT\$_GCR_6250) and is ignored.

¹Defined by the \$MTDEF macro

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

Table 6–6 (Cont.) Set Mode and Set Characteristics Magnetic Tape Characteristics

Characteristic ¹	Meaning
MT\$V_DENSITY MT\$\$_DENSITY	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 as follows: MT\$_DEFAULT Default system density. MT\$_GCR_6250 Group-coded recording, 6250 bpi. MT\$_PE_1600 Phase-encoded recording, 1600 bpi. MT\$_NRZI_800 Non-return-to-zero-inverted recording, 800 bpi. MT\$_BLK_833 Cartridge block mode recording ² .
MT\$V_FORMAT MT\$\$_FORMAT	Specifies the format in which all data transfers are performed. Possible format values are as follows: MT\$_DEFAULT Default system format. MT\$_NORMAL11 Normal PDP–11 format. Data bytes are recorded sequentially on tape with each byte occupying exactly one frame.

¹Defined by the \$MTDEF macro

²Only for the TK50, TZ30, and TF30

Table 6–7 Extended Device Characteristics for Tape Devices

Characteristic ¹	Meaning
MT2\$V_WBC_ENABLE	Enable write-back caching on a per unit basis.
MT2\$V_RDC_DISABLE	Disable read caching on a per unit basis.

¹Defined by the \$MT2DEF macro. Only for TU81-Plus drives.

Application programs that change specific magnetic tape characteristics should perform the following steps, as shown in Example 6–2 in Section 6.6:

- 1 Use the IO\$_SENSEMODE function to read the current characteristics.
- 2 Modify the characteristics.
- 3 Use the set mode function to write back the results.

Failure to follow this sequence will result in clearing any previously set characteristic.

Magnetic Tape Drivers

6.4 Magnetic Tape Function Codes

6.4.11 Data Security Erase

The data security erase function erases all data from the current position of the volume to 10 feet beyond the EOT reflective strip and then rewinds the tape to the BOT marker. It is a physical I/O function and requires the access privilege necessary to perform physical I/O functions. It is applicable only for the TA78, TU78, TA81, TK50, TU81, TU81-Plus, TZ30, and TF30 drives. The following function code is provided:

- IO\$_DSE

If the function is issued when a tape is positioned at the BOT marker, all data on the tape will be erased.

IO\$_DSE takes no device- or function-dependent arguments.

6.4.12 Pack Acknowledge

The pack acknowledge function sets the volume valid bit for all magnetic tape devices. It is a physical I/O function and requires the access privilege to perform physical I/O. The following function code is provided:

- IO\$_PACKACK

This function code takes no function-dependent arguments.

IO\$_PACKACK must be the first function issued when a volume is placed in a magnetic tape drive. IO\$_PACKACK is issued automatically when the DCL commands INITIALIZE or MOUNT are issued.

6.4.13 Available

The available function clears the volume valid bit for all magnetic tape drives, that is, it reverses the function performed by the pack acknowledge function (see Section 6.4.12). A rewind of the tape is performed (applicable to all tape drives). No unload function is issued to the drive. The following function code is provided:

- IO\$_AVAILABLE

This function takes no function-dependent arguments.

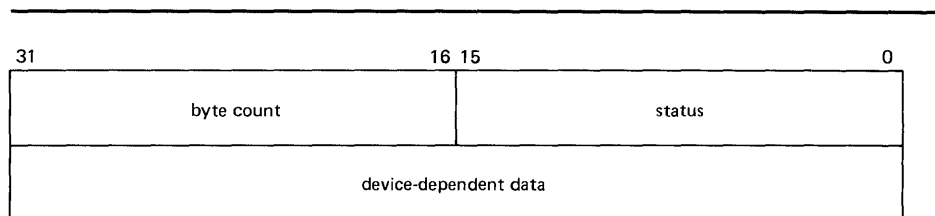
6.5 I/O Status Block

The I/O status block (IOSB) for QIO functions on magnetic tape devices is shown in Figure 6-6. Appendix A lists the status returns for these functions. (The *VMS System Messages and Recovery Procedures Reference Volume* provides explanations and suggested user actions for these returns.) Table 6-3 (in Section 6.3) lists the device-dependent data returned in the second longword. The IO\$_SENSEMODE function can be used to return that data.

Magnetic Tape Drivers

6.5 I/O Status Block

Figure 6–6 IOSB Contents



ZK-675-82

The byte count is the actual number of bytes transferred to or from the process buffer or the number of files or blocks skipped. (If a `IO$_SKIPRECORD` function is terminated by the detection of a tape mark, the count returned in the IOSB is a signed number reflecting the number of blocks skipped, plus 1 (forward tape motion) or minus 1 (reverse tape motion).

6.6 Programming Examples

The following program (Example 6–1) 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 a magnetic tape driver. However, this would have involved additional programming such as writing header labels and trailer labels.

Magnetic Tape Drivers

6.6 Programming Examples

Example 6-1 Magnetic Tape Program Example

```
*****
;
;
; .TITLE  MAGTAPE PROGRAMMING EXAMPLE
; .IDENT  /01/
;
; Define necessary symbols.
;
; $FIBDEF                                ;Define file information block
;                                         ;symbols
; $IODEF                                  ;Define I/O function codes
;
; Allocate storage for the necessary data structures.
;
;
; Allocate magtape device name string and descriptor.
;
TAPENAME:                                ;
; .LONG  20$-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                                ;Initialize buffer to
; .ASCII  /A/                               ;contain 'A'
; .ENDR                                     ;
;
; Now define the file information block (FIB), which the ACP uses
; in accessing and deaccessing the file.  Both the user and the ACP
; supply the information required in the FIB to perform these
; functions.
;
```

Example 6-1 Cont'd. on next page

Magnetic Tape Drivers

6.6 Programming Examples

Example 6-1 (Cont.) Magnetic Tape Program Example

```
FIB_DESCR:                                ;Start of FIB
      .LONG  ENDFIB-FIB                    ;Length of FIB
      .LONG  FIB                            ;Address of FIB
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
;
; 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
;
; *****
;
;                               Start Program
;
; *****
;
; The program first assigns a channel to the magnetic tape unit and
; then performs an access function to create and access a file called
; MYDATA.DAT. Next, the program writes 26 blocks of data (the letters
; of the alphabet) to the tape. The first block contains all A's, the
; next, all B's, and so forth. The program starts by writing a block of
; 256 bytes, that is, the block of A's. Each subsequent block is reduced
; in size by two bytes so that by the time the block of Z's is written,
; the size is only 206 bytes. The magtape ACP does not allow the reading
; of a file that has been written until one of three events occurs:
; 1. The file is deaccessed.
; 2. The file is rewound.
; 3. The file is backspaced.
; In this example the file is backspaced zero blocks and then read in
; reverse (incrementing the block size every block); the data is
; checked against the data that is supposed to be there. If no data
; errors are detected, the file is deaccessed and the program exits.
;
      .ENTRY  MAGTAPE_EXAMPLE, ^M<R3,R4,R5,R6,R7,R8>
;
; First, assign a channel to the tape unit.
;
      $ASSIGN_S TAPENAME,TAPECHAN          ;Assign tape unit
      CMPW     #SS$_NORMAL,RO              ;Success?
      BSBW     ERRCHECK                     ;Find out
;
; Now create and access the file MYDATA.DAT.
;
```

Example 6-1 Cont'd. on next page

Magnetic Tape Drivers

6.6 Programming Examples

Example 6-1 (Cont.) Magnetic Tape Program Example

```

$QIOW_S CHAN=TAPECHAN,-           ;Channel is magtape
        FUNC=#IO$_CREATE!IO$_ACCESS!IO$_CREATE,-;Function
        -                           ;is create
        IOSB=IOSTATUS,-           ;Address of I/O status
        -                           ;word
        P1=FIB_DESCR,-            ;FIB descriptor
        P2=#NAME_DESCR            ;Name descriptor
CMPW    #SS$_NORMAL,RO            ;Success?
BSBW    ERRCHECK                  ;Find out

;
; LOOP1 consists of writing the alphabet to the tape (see previous
; description).
;
        MOVL    #26,R5              ;Set up loop count
        MOVL    #256,R3             ;Set up initial byte count
; in R3
LOOP1:  ;Start of loop
        $QIOW_S CHAN=TAPECHAN,-     ;Perform QIOW to tape channel
        FUNC=#IO$_WRITEVBLK,-      ;Function is write virtual
        -                           ;block
        P1=BUFFER,-                ;Buffer address
        P2=R3                       ;Byte count
CMPW    #SS$_NORMAL,RO            ;Success?
BSBW    ERRCHECK                  ;Find out

;
; Now decrement the byte count in preparation for the next write
; operation and set up a loop count for updating the character
; written; 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

;
; The alphabet is now complete. Fall through LOOP1 and update the
; byte count so that it reflects the actual size of the last block
; written to tape.
;
        ADDL2   #2,R3               ;Update byte count

;
; The tape is now read, but first the program must perform one of
; the three functions described previously before the ACP allows
; read access. The program performs an ACP control function,
; specifying skip zero blocks. This is a special case of skip reverse
; and causes the ACP to allow read access.
;

```

Example 6-1 Cont'd. on next page

Magnetic Tape Drivers

6.6 Programming Examples

Example 6-1 (Cont.) Magnetic Tape Program Example

```
CLRL    FIB+FIB$L_CNTRLVAL    ;Set up to space zero blocks
MOVW    #FIB$C_SPACE,FIB+FIB$W_CNTRLFUNC ;Set up for space
;function
$QIOW_S CHAN=TAPECHAN,-      ;Perform QIOW to tape channel
        FUNC=#IO$_ACPCONTROL,- ;Perform an ACP control
        -                    ;function
        P1=FIB_DESCR        ;Define the FIB
CMPW    #SS$_NORMAL,RO       ;Success?
BSBW    ERRCHECK             ;Find out

;
; Read the file in reverse.
;

        MOVL    #26,R5        ;Set up loop count
        MOVB    #^A/Z/,R6     ;Get first character in R6
LOOP3:
        MOVAL   BUFFER,R7     ;And buffer address to R7
        $QIOW_S CHAN=TAPECHAN,- ;Channel is magtape
        FUNC=#IO$_READVBLK!IO$_REVERSE,- ;Function is read
        -                    ;reverse
        IOSB=IOSTATUS,-      ;Define I/O status quadword
        P1=BUFFER,-          ;And buffer address
        P2=R3                ;R3 bytes
CMPW    #SS$_NORMAL,RO       ;Success?
BSBW    ERRCHECK             ;Find out

;
; Check the data read to verify that it matches the data written.
;

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

;
; Now deaccess the file.
;

        $QIOW_S CHAN=TAPECHAN,- ;Channel is magtape
        FUNC=#IO$_DEACCESS,-    ;Deaccess function
        IOSB=IOSTATUS          ;I/O status

;
; Deassign the channel and exit.
;

EXIT:   $DASSGN_S CHAN=TAPECHAN ;Deassign channel
        RET                    ;Exit

;
; If an error had been detected, a program would normally
; generate an error message here. But for this example the
; program simply exits.
;
```

Example 6-1 Cont'd. on next page

Magnetic Tape Drivers

6.6 Programming Examples

Example 6–1 (Cont.) Magnetic Tape Program Example

```
MISMATCH:                                ;
      BRB      EXIT                       ;Exit
ERRCHECK:                                ;
      BNEQ     EXIT                       ;If not success, exit
      RSB      RSB                        ;Otherwise, return
      .END      MAGTAPE_EXAMPLE
```

The following example (Example 6–2) illustrates the recommended sequence for changing a device characteristic. Retrieve the current characteristics using a IO\$_SENSEMODE request, set the new characteristics bits, and then use IO\$_SETMODE to set the new characteristics.

Example 6–2 Device Characteristic Program Example

```
$QIOW_S -                                ; Get current characteristics.
      FUNC     = #IO$_SENSEMODE, -        ; - Sensemode
      CHAN     = CHANNEL, -              ; - Channel
      IOSB     = IO_STATUS, -            ; - IOSB
      P1       = BUFFER, -               ; - User buffer supplied
      P2       = #12                     ; - Buffer length = 12
      .
      .
      .
      (Check for errors)
      .
      .
      .
      (Set desired characteristics bits)
      .
      .
      .
      $QIOW_S -                            ; Set new characteristics.
      FUNC     = #IO$_SETMODE, -         ; - Set Mode
      CHAN     = CHANNEL, -              ; - Channel
      IOSB     = IO_STATUS, -            ; - IOSB
      P1       = BUFFER, -               ; - User buffer address
      P2       = #12                     ; - Buffer length = 12
      .
      .
      .
      (Check for errors)
      .
      .
      .
```

The following example (Example 6–3) shows ways of specifying sense mode and set mode, both with and without a user buffer specified, and with user buffers of different lengths.

Magnetic Tape Drivers

6.6 Programming Examples

Example 6-3 Set Mode and Sense Mode Program Example

```
.PSECT      IMPURE, NOEXE, NOSHR
$IODEF

DEVICE_NAME:      ; Name of device
.ASCID      /MUAO/ ;
CHANNEL:        ; VMS channel to device
.WORD      0      ;
BUFFER: .BLKL    3      ; Set/Sense characteristics
          ; buffer
IO_STATUS:      ; Final I/O status
.QUAD      0      ;

.PSECT      CODE, RD, NOWRT, EXE
.ENTRY      MAIN, ^M<>

$ASSIGN_S -      ; Assign a channel to device
  DEVNAM      = DEVICE_NAME, - ;
  CHAN        = CHANNEL, -     ;

BSBW  ERR_CHECK2      ; Check for errors

$QIOW_S -      ; Get current characteristics
  FUNC        = #IO$_SENSEMODE, - ; No user buffer supplied
  CHAN        = CHANNEL, -       ;
  IOSB        = IO_STATUS, -     ;

BSBW  ERR_CHECK      ; Check for errors

$QIOW_S -      ; Get current characteristics
  FUNC        = #IO$_SENSEMODE, - ; User buffer supplied, length
  CHAN        = CHANNEL, -       ; defaulted
  IOSB        = IO_STATUS, -     ;
  P1          = BUFFER, -       ;

BSBW  ERR_CHECK      ; Check for errors

$QIOW_S -      ; Get current characteristics
  FUNC        = #IO$_SENSEMODE, - ; User buffer supplied, length
  CHAN        = CHANNEL, -       ; = 8
  IOSB        = IO_STATUS, -     ;
  P1          = BUFFER, -       ;
  P2          = #8, -           ;

BSBW  ERR_CHECK      ; Check for errors

$QIOW_S -      ; Get extended characteristics
  FUNC        = #IO$_SENSEMODE, - ; User buffer supplied, length
  CHAN        = CHANNEL, -       ; = 12
  IOSB        = IO_STATUS, -     ;
  P1          = BUFFER, -       ;
  P2          = #12, -          ;

BSBW  ERR_CHECK      ; Check for errors
```

Example 6-3 Cont'd. on next page

Magnetic Tape Drivers

6.6 Programming Examples

Example 6-3 (Cont.) Set Mode and Sense Mode Program Example

```

$QIOW_S - ; Set new characteristics
  FUNC      = #IO$_SETMODE,- ; Length defaulted
  CHAN      = CHANNEL,- ;
  IOSB      = IO_STATUS,- ;
  P1        = BUFFER ;
BSBW ERR_CHECK ; Check for errors

$QIOW_S - ; Set new characteristics
  FUNC      = #IO$_SETMODE,- ; Length = 8
  CHAN      = CHANNEL,- ;
  IOSB      = IO_STATUS,- ;
  P1        = BUFFER,- ;
  P2        = #8 ;
BSBW ERR_CHECK ; Check for errors

$QIOW_S - ; Set extended characteristics
  FUNC      = #IO$_SETMODE,- ; Length = 12
  CHAN      = CHANNEL,- ;
  IOSB      = IO_STATUS,- ;
  P1        = BUFFER,- ;
  P2        = #12 ;
BSBW ERR_CHECK ; Check for errors

RET

.ENABLE LSB

ERR_CHECK:
  BLBS IO_STATUS,ERR_CHECK2 ; Continue if good IOSB
  MOVZWL IO_STATUS,-(SP) ; Otherwise, set up for stop
  BRB 10$ ; Branch to common code

ERR_CHECK2:
  BLBS R0,20$ ; Continue if good status
  PUSHL R0 ; Otherwise, set up for stop
10$: CALLS #1,G^LIB$STOP ; Stop execution
20$: RSB

.DISABLE LSB

.END MAIN

```

7 Mailbox Driver

The VMS operating system 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 much like other I/O devices, they are not hardware devices. Rather, mailboxes are a software-implemented ways to perform read and write operations.

Multiport memory mailboxes function in the same way as regular mailboxes. They can also be used by processes on different processors connected to an MA780 multiport memory option.

The *Guide to VMS Programming Resources* and the *VMS System Services Reference Manual* contain additional information on the use of mailboxes.

7.1 Mailbox Operations

Table 7–1 lists the different operations that software mailboxes perform.

Table 7–1 Mailbox Read and Write Operations

Operation	Description
Receive mail	A process initiates a read request to a mailbox to obtain data sent by another process. The process reads the data if a message was previously transmitted to the mailbox.
Receive notification of mail	A process specifies that it be notified through an AST when a message is sent to the mailbox.
Send mail (without notification of receipt)	A process initiates a write request to another mailbox to transmit data to second 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)	A process initiates a write request to another mailbox to transmit data to second process. The sending process waits until the receiving process reads the data before completing the I/O operation.
Reject mail	The receiving process reads messages from the mailbox, sorts out unwanted messages, and responds only to useful messages.

7.1.1 Creating Mailboxes

To create a mailbox and assign a channel and logical name to it, a process uses the Create Mailbox and Assign Channel (\$CREMBX) system service. The system enters the logical name in the job logical name table and gives it an equivalence name of MBAn, where *n* is a unique unit number.

Mailbox Driver

7.1 Mailbox Operations

`$CREMBX` also establishes the characteristics of the mailbox. These characteristics include a protection mask, permanence indicator, maximum message size, and buffer quota. A mailbox is created as either a temporary mailbox or a permanent mailbox; both types of mailboxes require privilege to create. Applications and restrictions on use of temporary and permanent mailboxes are described in the sections that follow. (See the *VMS System Services Reference Manual* for additional information on creating mailboxes.)

Other processes can assign additional channels to the mailbox using either `$CREMBX` or the Assign I/O Channel (`$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`.

If sufficient dynamic memory for the mailbox data structure is not available when a mailbox is created, a resource wait occurs if resource wait mode is enabled.

When a mailbox is created, a certain amount of space is specified for buffering messages that have been written to the mailbox, but they have not yet been read. The **bufquo** argument to the `$CREMBX` system service specifies this amount or quota. If that argument is omitted, its value defaults to the system generation parameter `DEFMBXBUFQUO`.

A message written to a mailbox, in the absence of an outstanding read request, is queued to the mailbox, and the size of the message (the `QIO P2` argument) is subtracted from the available buffering space. After the message is read, it is added back to the available buffering space.

If a process attempts to write to a mailbox that is full or has insufficient buffering space, and if the process has resource wait enabled (which is the default case), the process is placed in miscellaneous resource wait mode until sufficient space is available in the mailbox. If resource wait is not enabled, the I/O completes with the status return `SS$_MBFULL` in the I/O status block (`IOSB`).

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

7.1.2 Deleting Mailboxes

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 1. The system maintains a count of all channels and automatically deletes the mailbox when no more channels are assigned to it (that is, when the channel count reaches 0).

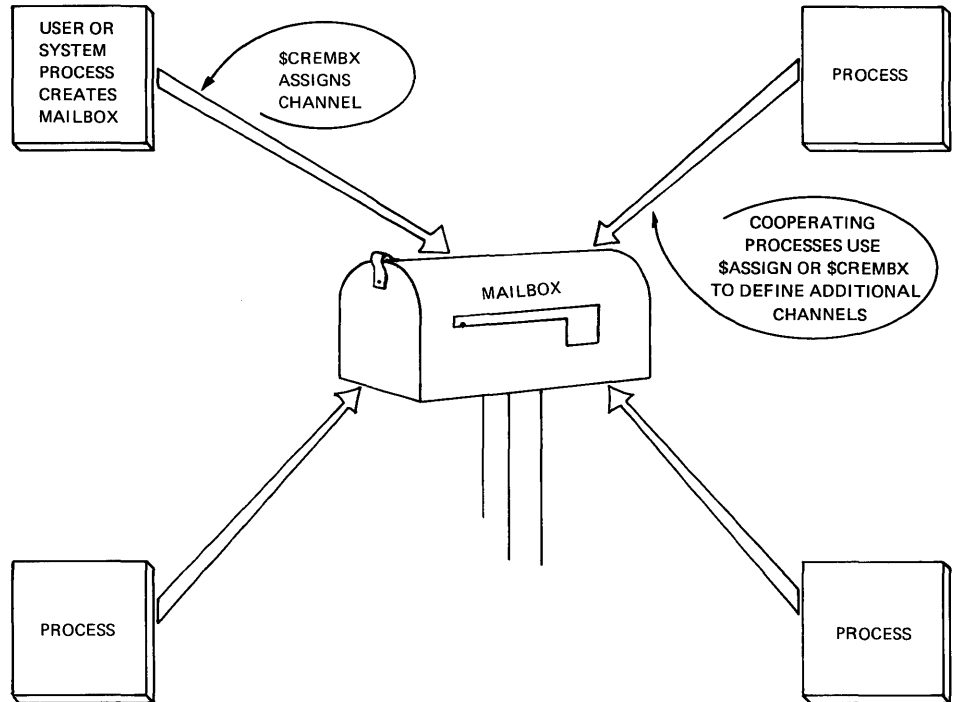
If a mailbox channel is deassigned, all messages sent through that channel are deleted unless the `IO$_M_NOW` function modifier was specified with the write request.

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

Mailbox Driver

7.1 Mailbox Operations

Figure 7–1 Multiple Mailbox Channels



ZK-676-82

When a temporary mailbox is deleted, its message buffer quota is returned to the process that created it. (No quota charge is made for permanent mailboxes.)

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 8–2 in Section 8.2.4.

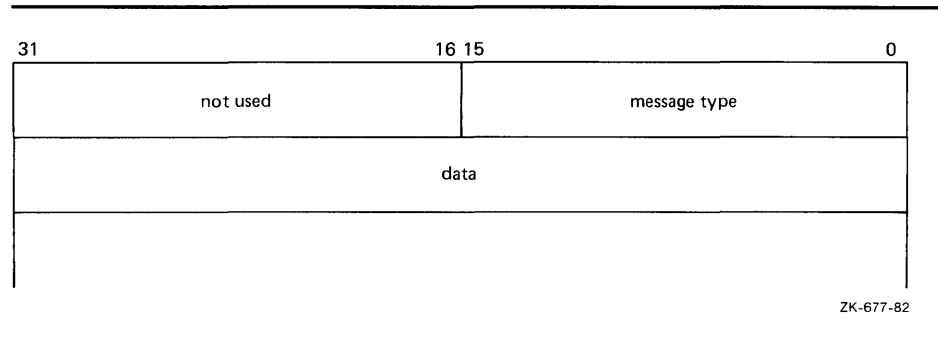
7.1.4 Mailbox Protection

Mailboxes (both temporary and permanent) are protected by a code, or mask, that is similar to the code used in protecting volumes. As with volumes, four types of users (defined by UIC) can gain access to a mailbox: SYSTEM, OWNER, GROUP, and WORLD. However, only three types of access—logical I/O, read, and write—are meaningful to users of a mailbox. Thus, when creating a mailbox, you can specify logical I/O, read, and write access to the mailbox separately for each type of user. Logical I/O access is required for any mailbox operation. The set protection function modifier provides additional control of mailbox access (see Section 7.3.5).

Mailbox Driver

7.1 Mailbox Operations

Figure 7–2 Typical Mailbox Message Format



Because temporary mailboxes are customarily used for interprocess communication between cooperating processes with the same group number, they are used more frequently than permanent mailboxes. The logical names of these mailboxes are entered in the group logical name table. Temporary mailboxes thus have two layers of protection. First, easy access to the logical names of temporary mailboxes is granted only to users who have the same group number as the creator of the mailbox; other users have no such easy access. Second, through the protection mask, the creator of the temporary mailbox grants additional security to the mailbox. As a rule, users who are not in the same group as the creator are excluded from using the mailbox.

Furthermore, the creator of a temporary mailbox can distinguish owners from other group members by granting read access and write access to a temporary mailbox—in addition to logical I/O access. For example, owners can be allowed only read access or only write access to the mailbox, but other members of the group can be allowed both read access and write access to the mailbox.

7.2 Device Information

You can obtain information on mailbox characteristics by using the Get Device/Volume Information (\$GETDVI) system service. (See the *VMS System Services Reference Manual*.)

\$GETDVI returns mailbox characteristics when you specify the item code DVI\$_DEVCHAR. Table 7–2 lists these characteristics, which are defined by the \$DEVDEF macro.

Mailbox Driver

7.2 Device Information

Table 7–2 Mailbox Characteristics

Characteristic ¹	Meaning
Dynamic Bits (Conditionally Set)	
DEV\$_SHR	Device is shareable.
DEV\$_AVL	Device is available.
Static Bits (Always Set)	
DEV\$_REC	Device is record-oriented.
DEV\$_IDV	Device is capable of input.
DEV\$_ODV	Device is capable of output.
DEV\$_MBX	Device is a mailbox.

¹Defined by the \$DEVDEF macro.

DVI\$_DEVCLASS and DVI\$_DEVTYPE return the device class and device type names, which are defined by the \$DCDEF macro. The device class for mailboxes is DC\$_MAILBOX. The device type is DT\$_MBX (or DT\$_SHRMBX if the mailbox is a shared memory mailbox). DVI\$_DEVBUFSIZ returns the buffer size, which is the maximum message size in bytes. DVI\$_DEVDEPEND returns a longword field in which the two low-order bytes contain the number of messages in the mailbox. (The two high-order bytes are not used and should be ignored.)

DVI\$_UNIT returns the mailbox unit number. Use of a mailbox to hold a termination message for a subprocess or a detached process requires that the parent process obtain this number to pass to the **mbxunt** argument of the \$CREPRC system service.

7.3 Mailbox Function Codes

The 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.

Mailbox Driver

7.3 Mailbox Function Codes

7.3.1 Read

Read mailbox functions are used to obtain messages written by other processes. The VMS operating system provides the following mailbox function codes:

- IO\$_READVBLK—Read virtual block
- IO\$_READLBLK—Read logical block
- IO\$_READPBLK—Read physical block

The following device- or function-dependent arguments are used with these codes:

- P1—The starting virtual address of the buffer that is to receive the message read. If P2 specifies a zero-length buffer, P1 is ignored.
- P2—The size of the buffer in bytes (limited by the maximum message size for the mailbox). A zero-length buffer may be specified. If a message longer than the buffer is read, the alternate success status SS\$_BUFFEROVF is returned in the I/O status block. In such cases, the message is truncated to fit the buffer. The driver does not provide a means for recovering the deleted portion of the message.

The following function modifier can be specified with a read request:

- IO\$_M_NOW—Complete the I/O operation immediately with no wait for a write request from another process

Figure 7-3 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; the read request can either precede or follow the write request.

If process A issues a read request before process B issues a write request, one of two events can occur. If process A did not specify the function modifier IO\$_M_NOW, process A's request is queued before process B issues 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 until process B issues the write request, and no data is transferred from process B to process A. In this case, the I/O status returned to process A is SS\$_ENDOFFILE.

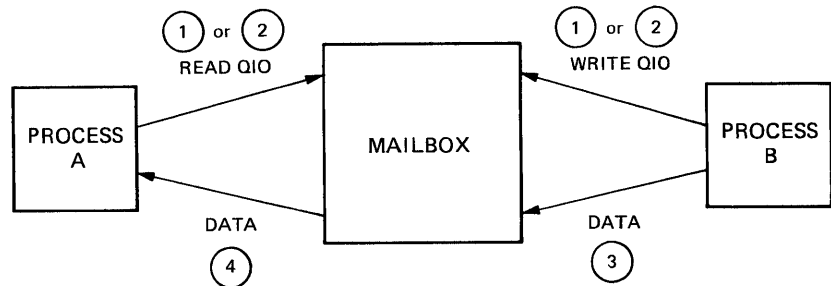
If process B sends a message (with no function modifier; see Section 7.3.2) before process A issues a read request (with or without a function modifier), process A finds a message in the mailbox. The data is transferred and the I/O operation is completed immediately.

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

Mailbox Driver

7.3 Mailbox Function Codes

Figure 7–3 Read Mailbox



NOTE: Numbers indicate order of events.

ZK-679-82

7.3.2 Write

Write mailbox functions are used to transfer data from a process to a mailbox. The VMS operating system provides the following mailbox function codes:

- IO\$_WRITEVBLK—Write virtual block
- IO\$_WRITELBLK—Write logical block
- IO\$_WRITEPBLK—Write physical block

These function codes take the following device- or function-dependent arguments:

- P1—The starting virtual address of the buffer that contains the message being written. If P2 specifies a zero-length buffer, P1 is ignored.
- P2—The size of the buffer in bytes (limited by the maximum message size for the mailbox). A zero-length buffer produces a zero-length message to be read by the mailbox reader.

The following function modifiers can be specified with a write request:

- IO\$_M_NOW—Complete the I/O operation immediately with no wait for another process to read the mailbox message
- IO\$_M_NORSWAIT—If the mailbox is full, the I/O operation fails with a status return of SS\$_MBFULL rather than placing the process in resource wait mode

Figure 7–4 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, a mailbox write request requires a corresponding mailbox read request (unless an error occurs), and the requests can be made in any sequence.

If process A issues a write request before process B issues a read request, one of two events can occur. If process A did not specify the function modifier IO\$_M_NOW, process A's write request is queued before process B issues 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

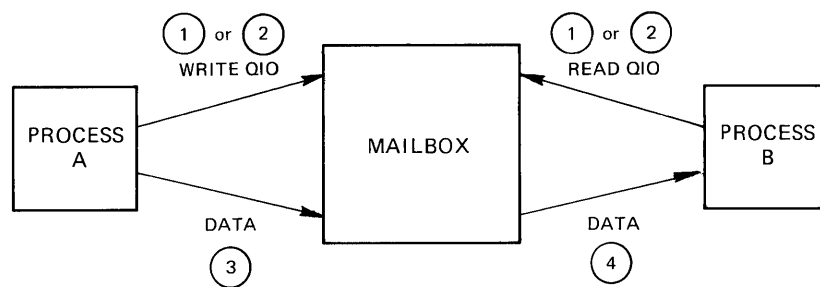
7.3 Mailbox Function Codes

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 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 function codes; all perform the same operation.

Figure 7-4 Write Mailbox



NOTE: Numbers indicate order of events.

ZK-680-82

7.3.3 Write End-of-File Message

Write end-of-file message functions 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. The VMS operating system provides the following function code:

- `IO$_WRITEOF`—Write end-of-file message

The following function modifier can be specified with a write end-of-file request:

- `IO$M_NOW`—Complete the I/O operation immediately

7.3.4 Set Attention AST

Set attention AST functions are used to specify that an AST be delivered to the requesting process when a cooperating process places an unsolicited read or write request in a designated mailbox. If a message exists in the mailbox when a request to enable a write attention AST is issued, the AST routine is activated immediately. If no message exists, the AST is delivered when a read or write message arrives. Thus the requesting process need not repeatedly check the mailbox status. You must have both logical I/O and read access to the mailbox prior to performing a set attention AST function.

The VMS operating system provides the following function codes:

- IO\$_SETMODE!IO\$_M_READATTN—Read attention AST
- IO\$_SETMODE!IO\$_M_WRTATTN—Write attention AST

These function codes take the following device- or function-dependent arguments:

- P1—AST address (request notification is disabled if the address is 0)
- 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 enabled only once; they must be explicitly reenabled after the AST has been delivered if you desire notification of the next unsolicited request. Both types of enable functions, and more than one of each type, can be set at the same time. The number of enable functions is limited only by the AST quota for the process.

Figure 7-5 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 might find the mailbox empty. If this request does not include the function modifier IO\$_M_NOW, it is queued before the next message arrives in the mailbox.

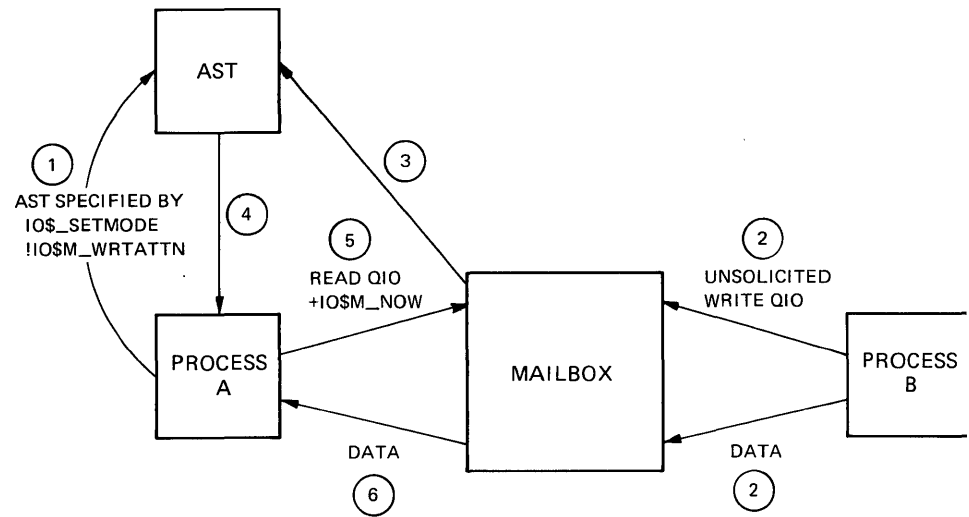
Figure 7-6 illustrates the read attention AST function. In this figure, an AST is set to notify process A when process B issues a read request for which no message is available.

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.

Mailbox Driver

7.3 Mailbox Function Codes

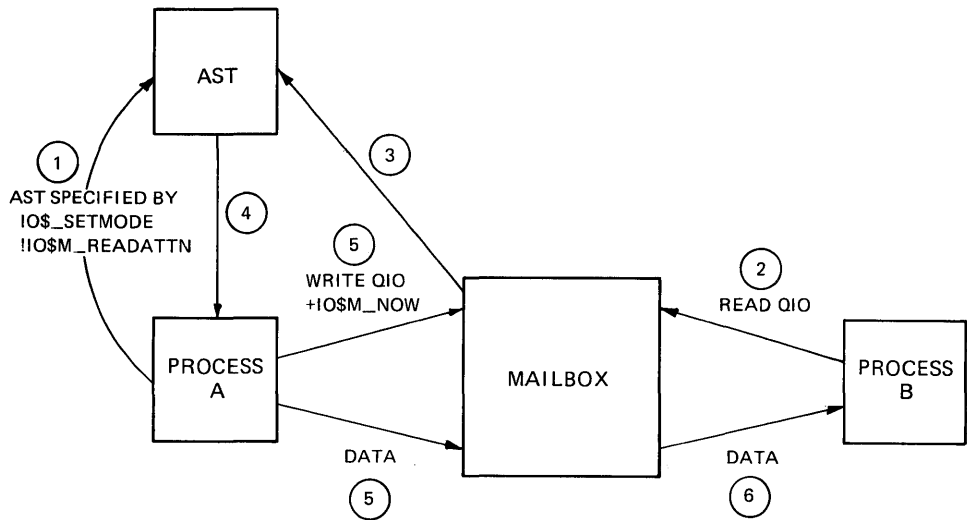
Figure 7-5 Write Attention AST (Read Unsolicited Data)



NOTE: Numbers indicate order of events.

ZK-681-82

Figure 7-6 Read Attention AST



NOTE: Numbers indicate order of events.

ZK-682-82

If several requesting processes set ASTs for read requests for 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.

7.3.5 Set Protection

Set protection functions allow the user to set volume protection on a mailbox (see Section 7.1.4). The requester must either be the owner of the mailbox or have BYPASS privilege. The VMS operating system provides the following function code:

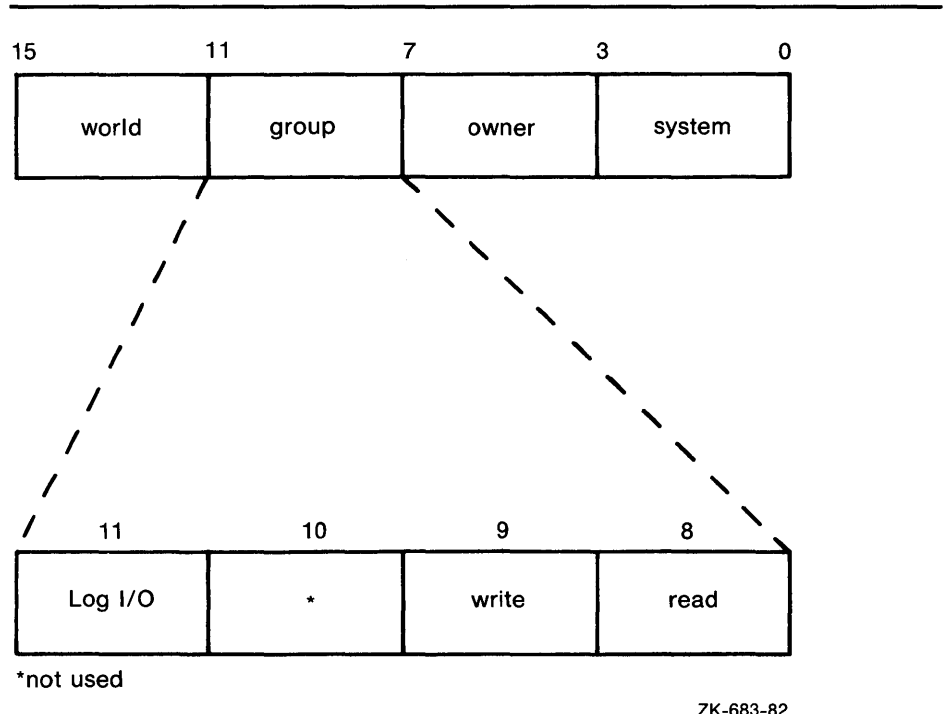
- IO\$_SETMODE!!IO\$_M_SETPROT—Set protection

This function code takes the following device- or function-dependent argument:

- P2—A volume protection mask

The protection mask specified by P2 is a 16-bit mask with four bits for each class of owner: SYSTEM, OWNER, GROUP, and WORLD, as shown in Figure 7-7.

Figure 7-7 Protection Mask



Only logical I/O, read, and write functions have meaning for mailboxes. A clear (0) bit implies that access is allowed. If P2 is 0 or unspecified, the mask is set to allow all read, write, and logical operations.

The I/O status block for the set protection function (see Figure 7-10) returns SS\$_NORMAL in the first word if the request was successful. If the request was not successful, the \$QIO system service returns SS\$_NOPRIV and both longwords of the I/O status block are returned as zeros.

Mailbox Driver

7.3 Mailbox Function Codes

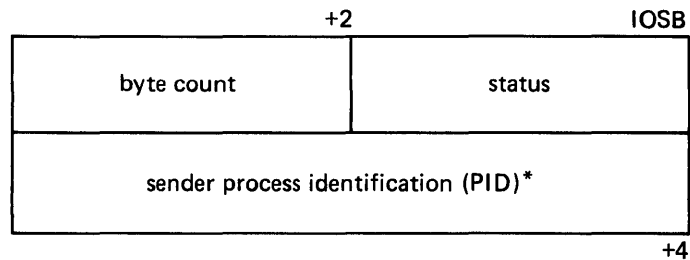
The set protection function is typically used when you want to inhibit write access and thus close off input to the mailbox. The mailbox can then be emptied without concern that a write operation might coincide with the read operation.

7.4 I/O Status Block

The I/O status blocks (IOSB) for mailbox read, write, and set protection QIO functions are shown in Figures 7-8, 7-9, and 7-10.

Appendix A lists the I/O status returns for these functions. In addition to these returns, the system services status returns `SS$_ACCVIO`, `SS$_INSFMEM`, `SS$_MBFULL`, `SS$_MBTOOSML`, and `SS$_NOPRIV` can be returned in R0. (The *VMS System Messages and Recovery Procedures Reference Volume* provides explanations and suggested user actions for both types of returns.)

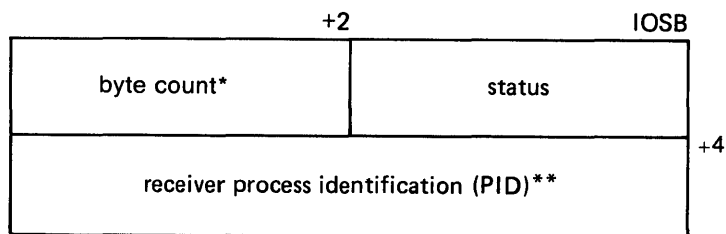
Figure 7-8 IOSB Contents - Read Function



*0 if the sender was a system process

ZK-684-82

Figure 7-9 IOSB Contents - Write Function



*equals P2 buffer size if successful request

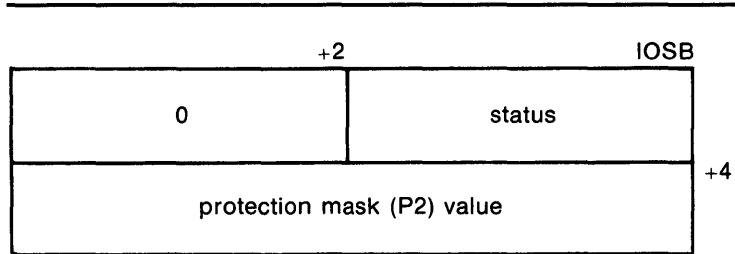
**0 if `IO$_NOW` was specified

ZK-685-82

Mailbox Driver

7.5 Programming Example

Figure 7–10 IOSB Contents - Set Protection Function



ZK-1201-82

7.5 Programming Example

The following program (Example 7–1) creates a mailbox and puts mail into it; no matching read is pending on the mailbox. First, the program illustrates that if the function modifier `IO$M_NOW` is not used when mail is deposited, the write function waits until a read operation is performed. In this case, `IO$M_NOW` is specified and the program continues after the mail is left in the mailbox.

Next, the mailbox is read. If there is no mail in the mailbox, the program waits because `IO$M_NOW` is not specified. `IO$M_NOW` should be specified if there is any doubt about 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 the data that 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

7.5 Programming Example

Example 7-1 Mailbox Driver Program Example

```

; *****
;
;
; TITLE MAILBOX DRIVER PROGRAM EXAMPLE
; IDENT /01/
;
; Define necessary symbols.
;
; $IODEF ;Define I/O function codes
;
; Allocate storage for necessary data structures.
;
; Allocate output device name string and descriptor.
;
DEVICE_DESCR:
;
; .LONG 20$-10$ ;Length of name string
; .LONG 10$ ;Address of name string
10$: .ASCII /SYS$OUTPUT/ ;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
;
; 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

```

Example 7-1 Cont'd. on next page

Mailbox Driver

7.5 Programming Example

Example 7-1 (Cont.) Mailbox Driver Program Example

```

;
; Finally, allocate space for the I/O status quadword.
;
STATUS:
        .QUAD    1                ;I/O status quadword
;
; *****
;
;                               Start Program
;
; *****
;
; The program first creates a mailbox and assigns a channel to the
; process output device. Then a message is placed in the mailbox and
; a message is received from the mailbox (the same message). Finally,
; the program prints the contents of the mailbox on the process output
; device.
;
START:  .WORD    0                ;Entry mask
        $CREMBX_S CHAN=MAILBOX_CHANNEL,- ;Channel is the mailbox
        PROMSK=#^X0000,-         ;No protection
        BUFQUO=#^X0060,-         ;Buffer quota is hex 60
        LOGNAM=MAILBOX_NAME,-    ;Logical name descriptor
        MAXMSG=#^X0060           ;Maximum message is hex 60
        CMPW    #SS$_NORMAL,RO    ;Successful mailbox creation?
        BSBW    ERROR_CHECK       ;Find out
        $ASSIGN_S -              ;Assign channel
        DEVNAM=DEVICE_DESCR,-    ;Device descriptor
        CHAN=DEVICE_CHANNEL      ;Channel
        CMPW    #SS$_NORMAL,RO    ;Successful channel assign?
        BSBW    ERROR_CHECK       ;Find out
;
; The program now writes to the mailbox using a write request that
; includes the function modifier IO$M_NOW so that it need not wait for
; a read request to the mailbox before continuing to the next step in
; the program.
;
        $QIOW_S FUNC=#IO$_WRITEVBLK!IO$M_NOW,- ;Write message NOW
        CHAN=MAILBOX_CHANNEL,-    ;to the mailbox channel
        P1=OUT_BOX_BUFFER,-       ;Write buffer
        P2=#OUT_LENGTH           ;Buffer length
        CMPW    #SS$_NORMAL,RO    ;Successful write request?
        BSBW    ERROR_CHECK       ;Find out
;
; Read the mailbox.
;

```

Example 7-1 Cont'd. on next page

Mailbox Driver

7.5 Programming Example

Example 7-1 (Cont.) Mailbox Driver Program Example

```

    $QIOW_S FUNC=#IO$_READVBLK,- ;Read the message
        CHAN=MAILBOX_CHANNEL,- ;in the mailbox channel
        IOSB=STATUS,- ;Define status block to
        - ;receive message length
        P1=IN_BOX_BUFFER,- ;Read buffer
        P2=#IN_LENGTH ;Buffer length
    CMPW #SS$_NORMAL,RO ;Successful read request?
    BSBW ERROR_CHECK ;Find out

;
; The program now determines how much mail is in the mailbox (this
; information is in STATUS+2) and then prints the mailbox message on
; the process output device.
;
    MOVZWL STATUS+2,R2 ;Byte count into R2
    $QIOW_S FUNC=#IO$_WRITEVBLK,- ;Write function to the
        CHAN=DEVICE_CHANNEL,- ;output device channel
        P1=IN_BOX_BUFFER,- ;Address of buffer to write
        P2=R2,- ;How much to write
        P4=#32 ;Carriage control

;
; Finally, deassign the channel and exit.
;
EXIT: $DASSGN_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 at this point if an error was
; detected. However, this example program simply exits.
;
ERROR_CHECK:
    BNEQ EXIT ;System service failure, exit
    RSB ;Otherwise, return

    .END START
```

8

Terminal Driver

This chapter describes the use of the VMS terminal driver (TTDRIVER) and the LAT port driver (LTDRIVER). The terminal driver supports the asynchronous, serial line multiplexers listed in Table 8-1. The terminal driver also supports the console terminal. The LAT port driver accommodates I/O requests from application programs, for example to make connections to remote devices, such as a printer, on a server (see Section 8.4.4).

8.1 Supported Terminal Devices

In addition to the multiplexers listed in Table 8-1, the terminal driver supports serial line interfaces that are included as part of all VAX processors. At least one such interface is always provided and is used to attach the system console terminal. This interface does not allow the setting of multiple terminal speeds, parity, or any maintenance functions, with the exception of the interface included with the VAX 8200 processor. The terminal devices supported by the VMS operating system for this interface are included in Table 8-1.

The remote command terminal, used by the DCL command SET HOST, also makes use of the features listed in Section 8.2.

Table 8-1 Supported Terminal Devices

Terminal Interface	No. of Lines	Output Silo/DMA	Split Speed	Bus	International Modem Control
CXY08	8	Yes ¹ /Yes	Yes	Q-BUS	Full
CXA16	16	Yes ¹ /Yes	Yes	Q-BUS	No
CXB16	16	Yes ¹ /Yes	Yes	Q-BUS	No
DZQ11	4	No/No	Yes	Q-BUS	No
DZQ11-CR	4	No/No	Yes	Q-BUS	No
uVAX2000	4	No/No	Yes	None	No
DZV11	4	No/No	No	Q-BUS	No
DHQ11	8	Yes ¹ /Yes	Yes	Q-BUS	Full
DHU11	16	Yes/Yes	Yes	UNIBUS	Full
DHV11	8	No/Yes	Yes	Q-BUS	Full
DMB32	8	No/Yes	Yes	VAXBI	Full
DMF32	8	Yes/Yes (lines 0 and 1)	Yes (lines 0 and 1)	UNIBUS	Yes
DMZ32	24	Yes/Yes	Yes	UNIBUS	Full
DZ11	8/16	No/No	No	UNIBUS	No

¹Depends on whether the DHV or DHU mode is selected when the board is installed

Terminal Driver

8.1 Supported Terminal Devices

Table 8–1 (Cont.) Supported Terminal Devices

Terminal Interface	No. of Lines	Output Silo/DMA	Split Speed	Bus	International Modem Control
DZ32	8	No/No	Limited	UNIBUS	No
LAT	2	No/Yes	²	N/A	²
VAX 8200 serial lines	4	No/No	No ³	None	No

²Server dependent

³The VMS operating system always supports the first serial line as a console interface. The first serial line, and the remaining three serial lines, are also supported as user terminal interfaces at a maximum speed of 1200 baud in configurations that may include a LAT terminal interface but do not include other terminal interfaces.

8.2 Terminal Driver Features

The VMS terminal driver provides the following features:

- Input processing
 - Command line editing and command recall
 - Control characters and special keys
 - Input character validation (read verify)
 - American National Standard (ANSI) escape sequence detection
 - Type-ahead feature
 - Specifiable or default input terminators
 - Special operating modes, such as NOECHO and PASTHRU
- Output processing
 - Efficiency
 - Limited full-duplex operation
 - Formatted or unformatted output
- Dial-up support
 - Modem control
 - Hangup on logging out
 - Preservation of process across hangups
- Miscellaneous
 - Terminal/mailbox interaction
 - Autobaud detection
 - Out-of-band control character handling

8.2.1 Input Processing

The VMS terminal driver defines many terminal characteristics and read function modifiers, which provide a wide range of options to an application program. These options allow multiple levels of control over the terminal driver's input process, ranging from the default of command line editing that provides a highly flexible user interface, to the PASTHRU mode, which inhibits input process interpretation of data.

8.2.1.1 Command Line Editing and Command Recall

The terminal driver input process defines a bounded set of line editing functions. These functions are available through control keys on all keyboards, and through some special keys on certain keyboards as well. Cursor movement is provided in single-character increments (left arrow or CTRL/D, right arrow or CTRL/F), or in multicharacter increments, to beginning of the line (CTRL/H), or end of the line (CTRL/E). The terminal driver supports both insert character and overstrike character modes. The insert/overstrike mode is the terminal's default characteristic¹ at the beginning of a read operation, but it can be changed dynamically with the toggle insert/overstrike key (CTRL/A). Deletion of characters is supported in both word (CTRL/J or line feed), and to the beginning of the line (CTRL/U) increments.

When you use the terminal driver's editing functions, the following restrictions result:

- The cursor cannot be moved to a previous line after a line wrap.
- A character cannot be inserted if the insertion would force a line wrap or if a tab follows the current cursor position.
- A word cannot be deleted at the beginning of a line after a line wrap.
- The line editing function cannot be assigned to other keys.

Command recall, initiated by CTRL/B or the up arrow, returns the last line entered to the command line buffer. At this point, the line can be edited or reentered by pressing the RETURN key. DCL extends command recall to the last 20 commands by using the TRM\$M_TM_NORECALL modifier to disable the terminal driver's recall mechanism.

Any control key that is not defined by line editing is ignored. For application programs that require control key input but do not perform QIO functions with special read modifiers, the SET TERMINAL/NOLINE_EDIT DCL command restores most of the terminal driver behavior described under VMS Versions 3.0 through 3.7

¹ It is suggested that new users specify overstrike mode.

Terminal Driver

8.2 Terminal Driver Features

8.2.1.2 Control Characters and Special Keys

A control character is a character that controls action at the terminal rather than passing data to a process. An ASCII control character has a code between 0 and 31, and 127 (hexadecimal 0 through 1F, and 7F); that is, all normal control characters plus DELETE. (Table 1 in Appendix B lists the numeric values for all control characters.)

Some control characters are entered at the terminal by simultaneously pressing the CTRL key and a character key, such as CTRL/x. Table 8-2 lists the VMS terminal control characters. Control character echo strings (CTRL/C, CTRL/Y, CTRL/O, and CTRL/Z) can be changed on a systemwide basis (see the *VMS System Generation Utility Manual*). Special keys, such as RETURN, LINE FEED, and ESCAPE, are entered by pressing a single key.

Several of the control characters do not function as described if the SET TERMINAL/LINE_EDIT DCL command is not specified. See the *VMS DCL Dictionary* for information on line editing function keys and the SET TERMINAL command.

Table 8-2 Terminal Control Characters

Control Character	Meaning
Cancel (CTRL/C - F6 ¹)	<p>Gains the attention of the enabling process if the user program has enabled a CTRL/C AST. If a CTRL/C AST is not enabled, CTRL/C is converted to CTRL/Y (see Section 8.4.3.2).</p> <p>The terminal performs a carriage-return/line-feed combination (carriage return followed by a line feed), echoes CANCEL, and performs another carriage-return/line-feed combination. If the terminal has the ReGIS characteristic or if CTRL/Y is pressed, the cancel ReGIS escape sequence is sent.</p> <p>Additional consequences of CTRL/C are as follows:</p> <ul style="list-style-type: none">• The type-ahead buffer is emptied.• CTRL/S and CTRL/O are reset.• All queued and in-progress write operations and all in-progress read operations are successfully completed. The status return is SS\$_CONTROL_C, or SS\$_CONTROL_Y if CTRL/C is converted to CTRL/Y.
Delete character (DELETE)	Removes the last character entered from the input stream.

¹F6 on the LK201 is Interrupt/Cancel.

Terminal Driver

8.2 Terminal Driver Features

Table 8–2 (Cont.) Terminal Control Characters

Control Character	Meaning
	DELETE (decimal 127 or hexadecimal 7F) is ignored if there are currently no input characters. Hardcopy 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 non-DELETE character is entered). Terminals that have the TT\$M_SCOPE characteristic echo DELETE by removing the character.
Delete line (CTRL/U)	Purges current input data. When CTRL/U is entered before the end of a read operation, the current input line is deleted. (In the case of a line-wrap, CTRL/U deletes only a line at a time.) If line editing is enabled (SET TERMINAL/LINE_EDIT is specified), the data from the beginning of the line to the current cursor position is deleted.
Delete word (CTRL/J or F13) (Line feed)	Delete word before cursor. Word terminators are all control characters, space, comma, dash, period, and ! " # \$ & ' () + @ [\] ^ { ~ / : ; < > = ? (see Appendix B).
Discard output (CTRL/O)	Discards output. Action is immediate. All output is discarded until the next read operation, the next write operation with a IO\$M_CANCTRLLO modifier, or the receipt of the next CTRL/O. The terminal echoes OUTPUT OFF. The current write operation (if any) and write operations performed while CTRL/O is in effect are completed with a status return of SS\$_CONTRLO. A second CTRL/O, which reenables output, echoes OUTPUT ON. CTRL/C, CTRL/Y, and CTRL/T cancel CTRL/O.
End of line (CTRL/E)	Moves the cursor to the end of the line.
Exit (CTRL/Z or F10)	Echoes EXIT when CTRL/Z is entered as a read terminator. By convention, CTRL/Z constitutes end-of-file.

Terminal Driver

8.2 Terminal Driver Features

Table 8–2 (Cont.) Terminal Control Characters

Control Character	Meaning
Interrupt (CTRL/Y)	<p>CTRL/Y is a special interrupt or attention character that is used to invoke the command interpreter for a logged-in process. CTRL/Y can be enabled with an IO\$_M_CTRLYAST function modifier to a IO\$_SETCHAR or IO\$_SETMODE function code. The command interpreter's CTRL/Y AST handler always takes precedence over a user program's CTRL/Y AST handler.</p> <p>Entering CTRL/Y results in an AST to an enabled process to signify that the user entered CTRL/Y from the terminal. The terminal performs a carriage-return/line-feed combination, echoes INTERRUPT, and performs another carriage-return/line-feed combination if the AST and echo are enabled. CTRL/Y is ignored (and not echoed) if the process is not enabled for the AST.</p> <p>Additional consequences of CTRL/Y are as follows:</p> <ul style="list-style-type: none">• The type-ahead buffer is flushed.• CTRL/S and CTRL/O are reset.• All queued and in-progress write operations and all in-progress read operations are successfully completed with a 0 transfer count. The status return is SS\$_CONTROLY.• The cancel ReGIS escape sequence is sent.
Move cursor left (CTRL/D ←)	Moves the cursor one position to the left.
Move cursor right (CTRL/F →)	Moves the cursor one position to the right.
Move cursor to beginning of line (CTRL/H or F12) (Back space)	Moves the cursor to the beginning of the line.
Purge type ahead (CTRL/X)	Purges the type-ahead buffer and performs a CTRL/U operation. Action is immediate. If a read operation is in progress, the operation is equivalent to CTRL/U.
Recall (CTRL/B or up arrow)	Recalls last command entered. DCL extends recall to several commands.

Terminal Driver

8.2 Terminal Driver Features

Table 8–2 (Cont.) Terminal Control Characters

Control Character	Meaning
Redisplay input (CTRL/R)	Redisplays current input. When CTRL/R is entered during a read operation, a carriage-return/line-feed combination is echoed on the terminal, and the current contents of the input buffer are 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\$_M_NOECHO is set.
Restart output (CTRL/Q)	Controls data flow; used by terminals and the driver. Restarts data flow to and from a terminal if previously stopped by CTRL/S. The action occurs immediately with no echo. CTRL/Q is also used to solicit read operations. CTRL/Q is meaningless if the line does not have the characteristic TT\$_M_TTSYNC, the characteristic TT\$_M_READSYNC, or is not currently stopped by CTRL/S.
RET (RETURN)	If used during a read (input) operation, RET echoes a carriage-return/line-feed combination. All carriage returns are filled on terminals with TT\$_M_CRFILL specified.
Stop output (CTRL/S)	Controls data flow; used by both terminals and the terminal 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 either the TT\$_M_TTSYNC characteristic or the TT\$_M_READSYNC characteristic.
TAB (CTRL/I)	Tabs horizontally. Advances to the next tab stop on terminals with the characteristic TT\$_M_MECHTAB, but the terminal driver assumes tab stops on MODULO 8 (multiples of 8) cursor positions. On terminals without this characteristic, enough spaces are output to move the cursor to the next MODULO 8 position.
Status (CTRL/T)	Displays the current time. CTRL/T also displays the current node and user name, the name of the image that is running, and information about system resources that have been used during the current terminal session.
Toggle insert/overstrike (CTRL/A or F14)	Changes current edit mode from insert to overstrike, or from overstrike to insert. The default mode (as set with SET TERMINAL/LINE_EDIT) is reset at the beginning of each line.

8.2.1.3 Read Verify

The read verify instructions provided by the terminal driver allow validation of data as each character is entered. Invalid characters are not echoed and terminate the operation. The terminal driver does not support full function field processing. Large data entry applications should use the VAX FMS or VAX TDMS layered products, which support the entire data entry environment. Section 8.4.1.4 describes the supported primitives.

Terminal Driver

8.2 Terminal Driver Features

8.2.1.4 Escape and Control Sequences

Escape and control sequences provide additional terminal control not furnished by the control characters and special keys (see Section 8.2.1.2). Escape sequences are strings of two or more characters, beginning with the escape character (decimal 27 or hexadecimal 1B), which 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.

The set mode characteristic `TT$M_ESCAPE` (see Table 8-5) is used to specify that VMS terminal lines can generate valid escape sequences. Also, the read function modifier `IO$M_ESCAPE` allows any read operation to terminate on an escape sequence regardless of whether `TT$M_ESCAPE` is set. If either `TT$M_ESCAPE` or `IO$M_ESCAPE` 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; a read buffer can contain other characters that are not part of an escape sequence, but a complete escape sequence always terminates a read operation. The return information in the read buffer and the I/O status block includes the position and size of the terminating escape sequence in the data record (see Section 8.5).

Any escape sequence received from a 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. If `SS$_PARTESCAPE` is returned, the application program must issue enough single-character read requests, without timeout, to read the remaining characters in the escape sequence, while parsing the syntax of the rest of the escape sequence. Use of the `TRM$_ESCTRMOVR` item code prevents `SS$_PARTESCAPE` errors. No syntax integrity is guaranteed across read operations. Escape sequences are never echoed. Valid escape sequences take any of the following forms (hexadecimal notation):

`ESC <int>...<int> <fin>` (7-bit environment)

`CSI <int>...<int> <fin>` (8-bit environment)

The keywords in the escape sequences indicate the following:

- ESC** The ESC key, a byte (character) of 1B. This character introduces the escape sequence in a 7-bit environment.
- CSI** The Control Sequence Introducer, a byte (character) of 9B. This character introduces the escape sequence in a 8-bit environment.
- <int>** An "intermediate character" in the range of 20 to 2F. This range includes the space character and 15 punctuation marks. An escape sequence can contain any number of intermediate characters, or none.
- <fin>** A "final character" in the range of 30 to 7E. This range includes uppercase and lowercase letters, numbers, and 13 punctuation marks.

Three additional escape sequence forms are as follows:

`ESC <;> <20-2F>...<30-7E>`

`ESC <?> <20-2F>...<30-7E>`

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

Terminal Driver

8.2 Terminal Driver Features

Control sequences, as defined by the ANSI standard, are escape sequences that include control parameters. Control sequences have the following format:

ESC [<par>...<par> <int>...<int> <fin> (7-bit environment)

CSI <par>...<par> <int>...<int> <fin> (8-bit environment)

The keywords in the escape sequences indicate the following:

- ESC The ESC key, a byte (character) of 1B.
- [A control sequence, a byte (character) of 5B.
- CSI The Control Sequence Introducer, a byte (character) of 9B.
- <par> A parameter specifier in the range of 30 to 3F.
- <int> An "intermediate character" in the range of 20 to 2F.
- <fin> A "final character" in the range of 40 to 7E.

For example, the position cursor control sequence is ESC [Pl ; Pc H. Pl is the desired line position and Pc is the desired column position.

The user guides for the various terminals list valid escape and control sequences. For example, the *VT100 User Guide* lists the escape and control sequences for VT100 terminals.

Section 8.2.1.2 describes control character functions during escape sequences.

Table 2 in Appendix B lists the valid ANSI and DIGITAL-private escape sequences for terminals that have the TT2\$M_ANSICRT, TT2\$M_DECCRT, TT2\$M_DECCRT2, TT2\$M_AVO, TT2\$M_EDIT, and TT2\$M_BLOCK characteristics (see Table 8-6). Table 2 in Appendix B also lists assumed and selectable ANSI modes and selectable DIGITAL-private modes. Only the names of the escape sequences and modes are listed (for more information see the specific user guide for any of the various terminals). Unless otherwise noted, the operation of escape sequences and modes is identical to the particular terminals that implement these features.

8.2.1.5 Type-Ahead Feature

Input (data received) from a VMS terminal is always independent of concurrent output (data sent) to a terminal. This feature 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 Table 8-5 and Section 8.4.3).

Data entered 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 the 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 Table 8-7).

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 Table 8-5 and Section 8.4.3). If TT\$M_HOSTSYNC is not set, CTRL/G (BELL) is returned to inform you that the type-ahead buffer is full. If characters are entered when the

Terminal Driver

8.2 Terminal Driver Features

type-ahead buffer is full, the next read operation completes with a status return of `SS$_DATAOVERUN`. If `TT$_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 begin eight characters before the type-ahead buffer fills unless the `TT2$_M_ALTTYPEAHD` characteristic is set. In that case, the system generation parameter `TTY_ALTALARM` is used. The driver sends a `CTRL/Q` to restart transmission when the type-ahead buffer empties completely.

The type-ahead buffer length is variable, with possible values in the range from 0 through 32,767. The length can be set on a systemwide basis through use of the system generation parameter `TTY_TYPAHDSZ`. Terminal lines that do a large amount of bulk input should use the characteristic `TT2$_M_ALTTYPEAHD`, which allows the use of a larger type-ahead buffer specified by the system generation parameters `TTY_ALTYPAMD` and `TTY_ALTALARM`. (`TTY_ALTYPAMD` specifies the total size of the alternate type-ahead buffer; `TTY_ALTALARM` specifies the threshold at which a `CTRL/S` is sent.)

Certain input-intensive applications, such as block mode input terminals, can take advantage of an optimization in the driver. If a terminal has the characteristic `TT2$_M_PASTHRU` and the read function modifier `IO$_M_NOECHO` is specified, data is placed directly into the read buffer and thereby eliminates the overhead for moving the data from the type-ahead buffer.

8.2.1.6 Line Terminators

A line terminator is the control sequence that you type at the terminal to indicate the end of an input line. Optionally, the application 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`, `LF`, `TAB`, or `BS` (see Appendix B). If line editing is enabled, the only terminators are `CR`, `CTRL/Z`, or an escape sequence. Control keys that do not have an editing function are nonfunctioning keys. If included in the request, the argument is a user-selected group of characters (see Section 8.4.1.2).

All characters are 7-bit ASCII characters unless data is input on an 8-bit terminal (see Section 8.4.1). The characteristic `TT$_M_EIGHTBIT` determines whether a terminal uses the 7-bit or 8-bit character set; see Table 8-5. All input characters (except some special keys; see Section 8.2.1.2) are tested against the selected terminators. The input is terminated when a match occurs or your input buffer fills.

The terminal driver notifies the job controller to initiate login when it detects a carriage return terminator on a line with no current process (provided the line is not a secure server or the type-ahead feature has not been disabled). A bell character is sent when the notification occurs. A notification character other than the bell character may be specified by setting the system generation parameter `TTY_AUTOCHAR`.

8.2.1.7 Special Operating Modes

The VMS terminal driver supports many special operating modes for terminal lines. (Tables 8-5 and 8-6 in Section 8.3 list these modes.) All special modes are enabled or disabled by the `set mode` and `set characteristics` functions (see Section 8.4.3).

8.2.2 Output Processing

Output handling is designed to be very efficient in the terminal driver. For example, on multiplexers that support both silo and direct memory access (DMA) output, the driver considers record size to decide dynamically which mode will result in the least overhead. The block size specified by the system generation parameter `TTY_DMASIZE` is the minimum size block that can be used in a DMA operation.

8.2.2.1 Duplex Modes

The VMS terminal driver can execute in either half- or full-duplex mode. These modes describe the terminal driver software, specifically the ordering algorithms used to service read and write requests, not the terminal communication lines.

In half-duplex mode, all read and write requests are inserted onto one queue. The terminal driver removes requests from the head of this queue and executes them one at a time; all requests are executed sequentially in the order in which they were issued.

In full-duplex mode, read requests (and all other requests except write requests) are inserted onto one queue and write requests onto another. The existence of two queues allows the driver to recognize the presence of two requests, such as a read request and a write request at the same time. However, the driver does not execute the read request and the write request simultaneously. When it is ready to service a request, the driver decides which request—the read request or the write request—to process next.

In the VMS terminal driver, write requests usually have priority. A write request can interrupt a current, but inactive, read request. A read request is current when the terminal driver removes that request from the head of the read queue. In a read operation, the read request becomes active when the first input character is received and echoed. Once a read request becomes active, all write requests are queued until the read completes. However, during a read operation many write requests can be executed before the first input character is entered at the terminal. Terminal lines that have the `TT$M_NOECHO` characteristic, or read functions that include the `IO$M_NOECHO` function modifier, do not inhibit write operations in full-duplex mode.

If a write function specifies the `IO$M_BREAKTHRU` modifier, the write operation is not blocked, even by an active read operation. `IO$M_BREAKTHRU` does not change the order in which write operations are queued.

When all I/O requests are entered using the Queue I/O Request and Wait (`$QIOW`) system service, there can be only one current I/O request at a time. In this case, the order in which requests are serviced is the same for both half- and full-duplex modes.

The type-ahead buffer always buffers input data for which there is no current read request, in both half- and full-duplex modes.

Terminal Driver

8.2 Terminal Driver Features

8.2.2.2 Formatting of Output

By default, output data is subject to formatting by the terminal driver. This formatting includes actions such as wrapping, tab expansion, uppercase, and fallback conversions. Applications that do not require formatting of data can write with the `IO$_NOFORMAT` modifier and thereby reduce overhead. `IO$_NOFORMAT` overrides all formatting except fallback translation. Setting the `PASTHRU` mode (`TT2$_PASTHRU`) is equivalent to writing with the `noformat` modifier.

Fallback conversions occur regardless of formatting mode.

8.2.3 Dial-Up Support

The VMS operating system supports modem control (for example, Bell 103A, Bell 113, or equivalent) for all supported multiplexers in autoanswer, full-duplex mode. The terminal driver does not support half-duplex operations on modems such as the Bell 202. Also not supported are modems that use circuit 108/1 (connect data set to line signal) in place of the data terminal ready (DTR) signal. Most U.S. and European modems use the data terminal ready signal, which is the signal supported by the VMS operating system.

8.2.3.1 Modem Signal Control

Dial-up lines with the characteristic `TT$_MODEM` are monitored periodically to detect a change in the modem carrier signals data set ready (DSR), calling indicator (RING), or request to send (RTS). The system generation parameter `TTY_SCANDELTA` establishes the dial-up monitoring period for multiplexers that do not support modem signal transition interrupts (see Table 8-1).

If a line's carrier signal is lost, the driver waits two seconds for the carrier signal to return. If bit 0 of the system generation parameter `TTY_DIALTYPE` is set to 1, the driver does not wait. Bit 0 is 0 by default for countries with Bell System standards, but that bit should be set to 1 for countries with Comite Consultatif Internationale (CCITT) standards. If the carrier signal is not detected during this time, the line is hung up. The hangup action can signal 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 a minimum of two seconds after the hangup sequence begins. The hangup sequence is not reversible. If the line hangs up, all enabled `CTRL/Y` and out-of-band ASTs are delivered; the `CTRL/Y` AST P2 argument is overwritten with `SS$_HANGUP`. The I/O operation in progress is canceled, and the status value `SS$_HANGUP` is returned in the I/O status block. DCL is responsible for process deletion after `CTRL/Y` is delivered. If the process is suspended, DCL cannot run, and therefore deletion cannot occur, until the process is resumed.

For terminals with the `TT$_MODEM` characteristic, `TT$_REMOTE` reflects the state of the carrier signal. `TT$_REMOTE` is set when the carrier signal changes from off to on, and cleared when the carrier signal is lost.

A line that does not have `TT$_MODEM` set does not respond to modem signals or set the DTR signal. Modem signals can be set and sensed manually through use of the `IO$_MAINT` function modifier (see Section 8.4.3.3).

Terminal Driver

8.2 Terminal Driver Features

The VMS terminal driver default modem protocol meets the requirements of the United States and of European countries. This protocol is capable of working in automatic answer mode and can also perform manually dialed outgoing calls. The protocol supports the requirements of most known international telephone networks. Enhanced modem features are used on multiplexers that support them; processor polling is not necessary. The protocol also functions in a subset mode for the multiplexers that do not support full modem signals (see Table 8-1).

Table 8-3 lists the control and data signals used in a full modem control mode configuration (in a two-way simultaneous, symmetrical transmit mode). Figure 8-1 is a flowchart that shows a typical signal sequence for a terminal operation in this mode. The flowchart shows the states that the modem transition code goes through to detect different types of transitions in modem state. These transitions allow the driver to detect loss of lines that have been idle for several minutes. Modem states do not affect the ability of the system to transmit characters.

Set mode function modifiers are provided to allow a process to activate or deactivate modem control signals (see Section 8.4.3.3).

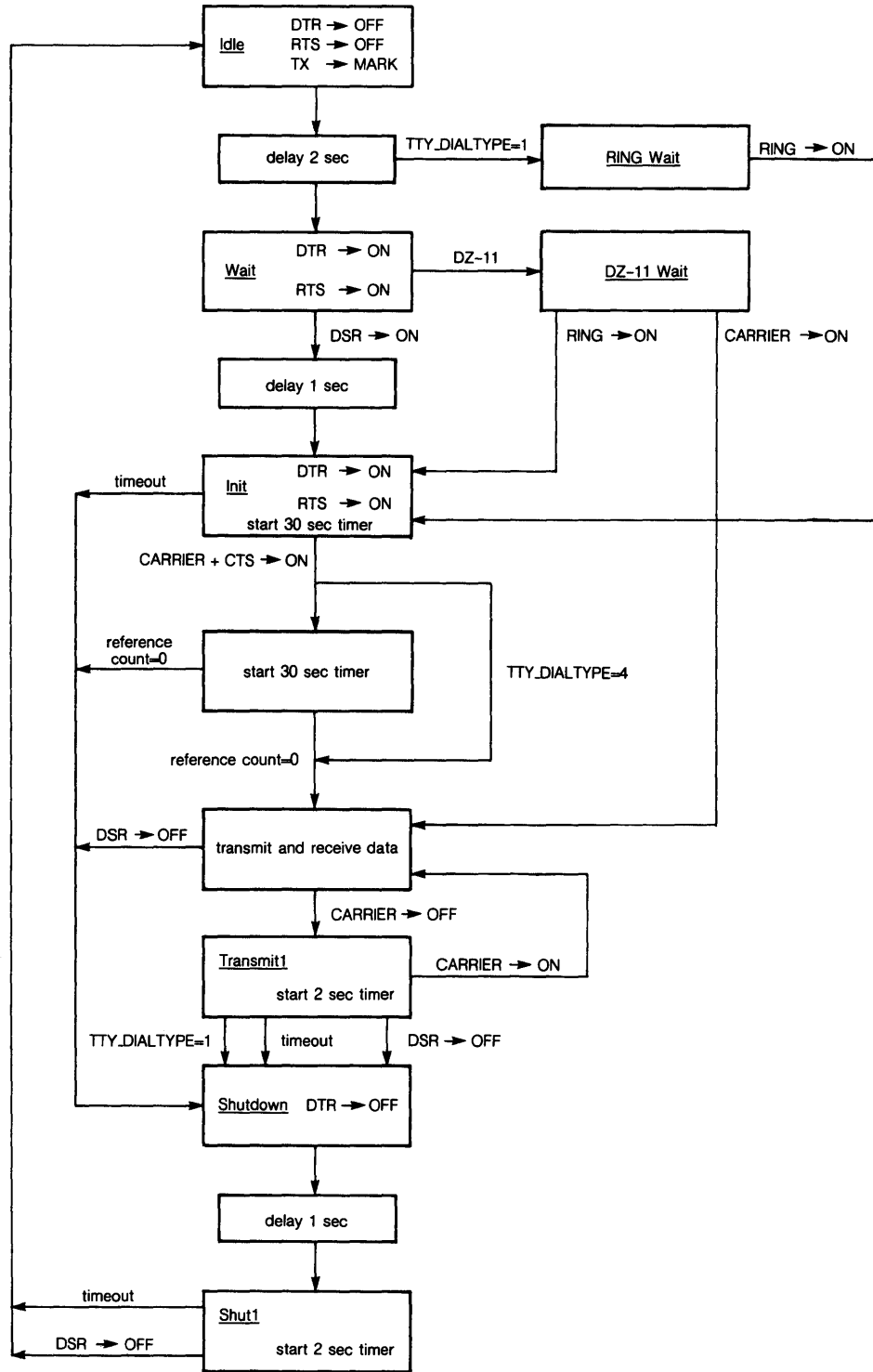
Bit 1 of the system generation parameter `TTY_DIALTYPE` enables alternate modem protocol on a systemwide basis. If bit 1 is 0 (the default), the RING signal is not used. If bit 1 is 1, the modem protocol delays setting the DTR signal until the RING signal is detected.

Remote terminal connections have a timeout feature for the security of dial-up lines. If no channel is assigned to the port within 30 seconds, or a port with an assigned channel is not allocated, the DTR signal is dropped. Such action prevents an unused terminal from tying up a line. However, there are configurations (such as a printer connected to a remote line) in which the line should not be dropped even though it is not being used interactively. To bypass the 30-second timeout, set the system generation parameter `TTY_DIALTYPE` to 4. (Note that if `TTY_DIALTYPE` is equal to 4, all dial-up lines will skip the timeout waiting for a channel to be assigned.)

Terminal Driver

8.2 Terminal Driver Features

Figure 8-1 Modem Control - Two-Way Simultaneous Operation



ZK-687-82

Terminal Driver

8.2 Terminal Driver Features

Table 8–3 Control and Data Signals (Full Modem Mode Configuration)

Signal	Source	MUX ¹	Meaning
Transmitted data (TxD)	Computer	All	The data originated by the computer and transmitted through the modem to one or more remote terminals.
Received data (RxD)	Modem	All	The data generated by the modem in response to telephone line signals received from a remote terminal and transferred to the computer.
Request to send (RTS)	Computer	Full	If present (ON condition), RTS directs the modem to assume the transmit mode. If not present (OFF condition), RTS directs the modem to assume the nontransmit mode after all transmit data has been transmitted.
Clear to send (CTS)	Modem	Full	Indicates whether the modem is ready (ON condition) or not ready (OFF condition) to transmit data on the telephone line.
Data set ready (DSR)	Modem	Full	If present (ON condition), DSR indicates that the modem is ready to transmit and receive; that is, the modem is connected to the line and is ready to exchange further control signals with the computer to initiate the exchange of data. If DSR is not present (OFF condition), the modem is not ready to transmit and receive. If DSR is detected, the VMS operating system initiates a 30-second timer. This ensures that the phone line will be disconnected if CARRIER is not detected.
Data channel received line signal detector (CARRIER)	Modem	All	If present (ON condition), CARRIER indicates that the received data channel line signal is within appropriate limits, as specified by the modem. If not present (OFF condition), the received signal is not within appropriate limits.

¹Multiplexers (All = any supported controller; Full = DZ32, DMF32, DMB32, DMZ32, DHU11, DHV11, and CXY08)

Terminal Driver

8.2 Terminal Driver Features

Table 8–3 (Cont.) Control and Data Signals (Full Modem Mode Configuration)

Signal	Source	MUX ¹	Meaning
Data terminal ready (DTR)	Computer	All	If present (ON condition), DTR indicates that the computer is ready to operate, prepares the modem to connect to the telephone line, and maintains the connection after it has been made by other means. DTR can be present whenever the computer is ready to transmit or receive data. If DTR is not present (OFF condition), the modem disconnects the modem from the line.
Calling indicator (RING)	Modem	All	Indicates whether a calling signal is being received by the modem. Bit 1 of the system generation parameter TTY_DIALTYPE must be set (=1). If RING is detected, the VMS operating system initiates a 30-second timer. This ensures that the phone line will be disconnected if CARRIER is not detected.

¹Multiplexers (All = any supported controller; Full = DZ32, DMF32, DMB32, DMZ32, DHU11, DHV11, and CXY08)

8.2.3.2 Hangup on Logging Out

By default, logging out on a line with modem signals will not break the connection. If `TT2$M_HANGUP` is set, modem signals are dropped when the process logs out. If `TT2$M_MODHANGUP` is set, no privilege is required to change the state of `TT2$M_HANGUP`. By setting `TT2M_HANGUP`, system managers can prevent nonprivileged users who are not logged in from tying up a dial-in line.

8.2.3.3 Preservation of a Process Across Hangups

Disconnectable terminals allow a connection to a physical terminal line to be broken without losing the job. The following `SYSGEN` command allows terminals to be disconnectable terminals:

```
SYSGEN> CONNECT VTA0/NOADAPTER/DRIVER=TTDRIVER
```

After this command is entered, a terminal with the `TT2$M_DISCONNECT` characteristic logs in as `VTAn:`, rather than with the physical terminal name. When a terminal is set up in this manner, no input or output operations are allowed to the physical device; I/O is automatically redirected to the appropriate virtual terminal.

Following are four ways in which a terminal can become disconnected:

- Modem signals between the host and the terminal are lost.

Terminal Driver

8.2 Terminal Driver Features

- A user presses the BREAK key on a terminal that has the TT2\$M_SECURE characteristic.
- A user issues the DCL command DISCONNECT.
- A user issues the DCL command CONNECT/CONTINUE.

After validated as a user, you can connect to a disconnected process in either of the following ways:

- Allow the login process to make the connection.
- Issue the DCL command CONNECT.

8.2.4 Terminal/Mailbox Interaction

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

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

Mailboxes used in this way carry status messages, not terminal data, from the driver to the user program. For example, when data is received from a terminal for which no read request is outstanding (unsolicited data), a message is sent to the associated mailbox to indicate data availability. On receiving this message, the user program reads the channel assigned to the terminal to obtain the data. Messages are sent to mailboxes under the following conditions:

- Unsolicited data in the type-ahead buffer. The use of the associated mailbox can be enabled and disabled as a subfunction of the read and write requests (see Sections 8.4.1 and 8.4.2). (Initially, mailbox messages are enabled on all terminals. This is the default state.) Thus, the user process can enter into a dialogue with the terminal after an unsolicited data message arrives. Then, after the dialogue is over, the user process can reenable the unsolicited data message function on the last I/O exchange. Only one message is sent between read operations.
- Terminal hangup. When a remote line loses the carrier signal, it hangs up; a message is sent to the mailbox. When hangup occurs on lines that have the characteristic TT\$M_REMOTE set, the line returns to local mode.
- Broadcast messages. If the characteristic TT2\$M_BRDCSTMBX is set, broadcasts sent to a terminal are placed in the mailbox (this is independent of the state of TT\$M_NOBRDCST).

Messages placed in the mailbox have the following content and format (see Figure 8-2):

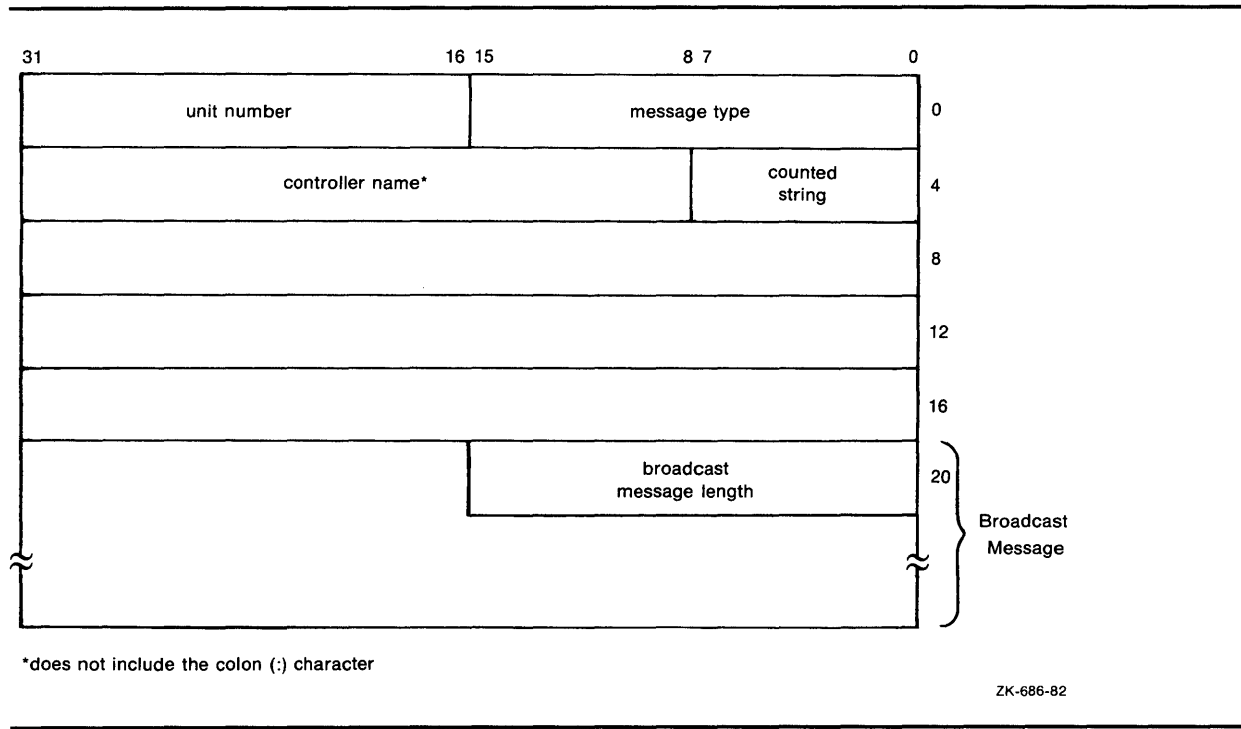
- Message type. The codes MSG\$_TRMUNSOLIC (unsolicited data), MSG\$_TRMHANGUP (hangup), and MSG\$_TRMBRDCST (terminal broadcast) identify the type of message. Message types are identified by the \$MSGDEF macro.
- Device unit number to identify the terminal that sent the message.

Terminal Driver

8.2 Terminal Driver Features

- Counted string to specify the device name.
- Controller name.
- Message (for broadcasts).

Figure 8–2 Terminal Mailbox Message Format



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. Chapter 7 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.

8.2.5 Autobaud Detection

If you specify the /AUTOBAUD qualifier with the SET TERMINAL command, automatic baud rate detection is enabled, allowing the terminal baud rate to be set when you log in. The baud rate is set at login by pressing the RETURN key two or more times separated by an interval of at least one second. (Pressing a key other than RETURN might detect the wrong baud rate; if this occurs, wait for the login procedure to time out before continuing.) The supported baud rates are 110, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 9600, and 19200. Parity is allowed on these lines.