OpenVMS I/O User's Reference Manual

December 1995

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

Revision/Update Information:	This manual supersedes the *OpenVMS I/O User's Reference Manual*, OpenVMS AXP Version 1.5, OpenVMS VAX Version 6.0
Software Version:	OpenVMS Alpha Version 7.0 OpenVMS VAX Version 7.0

Digital Equipment Corporation
Maynard, Massachusetts

December 1995

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Digital or an authorized sublicensor.

Digital conducts its business in a manner that conserves the environment and protects the safety and health of its employees, customers, and the community.

© Digital Equipment Corporation 1995. All rights reserved.

The following are trademarks of Digital Equipment Corporation: Alpha, AXP, Bookreader, DECwindows, Digital, OpenVMS, VAX, VAX DOCUMENT, VAXcluster, VMS, VMScluster, and the DIGITAL logo.

All other trademarks and registered trademarks are the property of their respective holders.

ZK6136

This document is available on CD–ROM.

Contents

Preface xvii

1 ACP–QIO Interface

1.1 ACP Functions and Encoding 1–2
1.2 File Information Block (FIB) 1–3
1.3 ACP Subfunctions 1–7
1.3.1 Directory Lookup 1–7
1.3.1.1 Input Parameters 1–7
1.3.1.2 Operation 1–8
1.3.1.3 Directory Entry Protection 1–9
1.3.2 Access 1–9
1.3.2.1 Input Parameters 1–9
1.3.2.2 Operation 1–10
1.3.3 Extend 1–10
1.3.3.1 Input Parameters 1–11
1.3.3.2 Operation 1–12
1.3.4 Truncate 1–12
1.3.4.1 Input Parameters 1–12
1.3.4.2 Operation 1–13
1.3.5 Read/Write Attributes 1–13
1.3.5.1 Input Parameters 1–14
1.4 ACP QIO Record Attributes Area 1–18
1.5 ACP–QIO Attributes Statistics Block 1–20
1.6 Major Functions 1–22
1.6.1 Create File 1–23
1.6.1.1 Input Parameters 1–23
1.6.1.2 Disk ACP Operation 1–24
1.6.1.3 Directory Entry Creation 1–25
1.6.1.4 Magnetic Tape ACP Operation 1–25
1.6.2 Access File 1–25
1.6.2.1 Input Parameters 1–26
1.6.2.2 Operation 1–26
1.6.3 Deaccess File 1–27
1.6.3.1 Input Parameters 1–27
1.6.3.2 Operation 1–27
1.6.4 Modify File 1–27
1.6.4.1 Input Parameters 1–28
1.6.4.2 Operation 1–28
1.6.5 Delete File 1–28
1.6.5.1 Operation 1–29

1.6.6 Movefile Subfunction 1–29
1.6.6.1 Calling the Movefile Subfunction 1–29
1.6.6.2 Input Parameters 1–30
1.6.6.3 Operation 1–31
1.6.7 Mount 1–32
1.6.8 ACP Control 1–32
1.6.8.1 Input Parameters 1–33
1.6.8.2 Magnetic Tape Control Functions 1–33
1.6.8.3 Miscellaneous Disk Control Functions 1–34
1.6.8.4 Disk Quotas 1–35
1.7 I/O Status Block 1–37

2 Disk Drivers

2.1 Supported Disk Devices and Controllers 2–1
2.1.1 UDA50 UNIBUS Disk Adapter 2–1
2.1.2 KDA50 Disk Controller 2–2
2.1.3 KDB50 Disk Controller 2–2
2.1.4 HSC40, HSC50, and HSC70 Controllers 2–2
2.1.5 SII Integral Adapter 2–3
2.1.6 KFQSA Adapter 2–3
2.1.7 RQDX3 Controller 2–3
2.1.8 RA70 and RA90 Disk Drives 2–3
2.1.9 RA60 Disk 2–3
2.1.10 RA80/RB80/RM80 and RA81 Fixed-Media Disks 2–3
2.1.11 RB02 and RL02 Cartridge Disk (VAX Only) 2–4
2.1.12 RC25 Disk (VAX Only) 2–4
2.1.13 RD53 and RD54 Disks (VAX Only) 2–4
2.1.14 RF30 and RF71 Disks 2–5
2.1.15 RK06 and RK07 Cartridge Disks (VAX Only) 2–5
2.1.16 RM03 and RM05 Pack Disks (VAX Only) 2–5
2.1.17 RP05 and RP06 Disk (VAX Only) 2–5
2.1.18 RP07 Fixed Media Disk (VAX Only) 2–5
2.1.19 RRD40 and RRD50 Read-Only Memory (CD–ROM) 2–5
2.1.20 RX01 Console Disk (VAX Only) 2–6
2.1.21 RX02 Disk (VAX Only) 2–6
2.1.22 RX23 (VAX Only) 2–7
2.1.23 RX33 (VAX Only) 2–7
2.1.24 RX50 (VAX Only) 2–7
2.1.25 RZ22, RZ23, and RZ55 Disks 2–7
2.1.26 TU58 Magnetic Tape (DECtape II) 2–7
2.2 Driver Features 2–8
2.2.1 Dual-Pathed Disks 2–8
2.2.2 Dual Porting MASSBUS Disks 2–9
2.2.2.1 Port Selection and Access Modes 2–9
2.2.2.2 Disk Use and Restrictions 2–10
2.2.2.3 Restriction on Dual-Ported Non-DSA Disks in a Cluster 2–10
2.2.3 Dual-Pathed DSA Disks 2–11
2.2.4 Dual-Porting HSC Disks 2–11
2.2.5 Dual-Pathed RF-Series Disks 2–11
2.2.6 Data Check 2–12
2.2.7 Effects of a Failure During an I/O Write Operation 2–13
2.2.8 Overlapped Seeks 2–13

2.2.9 Error Recovery 2–14
2.2.9.1 Skip Sectoring 2–14
2.2.10 Logical-to-Physical Translation (RX01 and RX02) 2–15
2.2.11 DIGITAL Storage Architecture (DSA) Devices 2–16
2.2.11.1 Bad Block Replacement and Forced Errors for DSA Disks 2–17
2.2.12 VAXstation 2000 and MicroVAX 2000 Disk Driver 2–18
2.2.13 SCSI Disk Class Driver 2–18
2.2.14 Audio Extensions to the SCSI Disk Class Driver 2–19
2.2.14.1 $QIO Interface to Audio Functionality of the SCSI Disk Class Driver 2–20
2.2.14.2 Defining an Audio Control Block (AUCB) 2–20
2.2.14.3 Error Handling in Applications Using SCSI Audio Functions 2–23
2.2.14.4 Using CD–ROM to Store Both Data and Audio Information 2–25
2.2.14.5 Programming Audio Applications 2–26
2.2.14.6 Application Program Example Using SCSI Audio Capabilities (VAX only) 2–26
2.3 Disk Driver Device Information 2–26
2.4 Disk Function Codes 2–28
2.4.1 Read 2–33
2.4.2 Write 2–34
2.4.3 Sense Mode 2–35
2.4.4 Set Density 2–35
2.4.5 Search 2–35
2.4.6 Pack Acknowledge 2–36
2.4.7 Unload 2–36
2.4.8 Available 2–36
2.4.9 Seek 2–36
2.4.10 Write Check 2–36
2.4.11 Set Preferred Path 2–37
2.4.11.1 Forcing a Path Change 2–38
2.4.11.2 Using IO$_SETPRFPATH with Disks Dual Pathed Between HSCs 2–38
2.4.11.3 Using IO$_SETPRFPATH with Disks Dual Pathed Between Systems 2–39
2.4.11.4 Using IO$_SETPRFPATH with Disks Accessed Through MSCP Servers 2–39
2.4.11.5 Using IO$_SETPRFPATH with Phase I Volume Shadowing 2–39
2.4.11.6 Using IO$_SETPRFPATH with Phase II Volume Shadowing 2–39
2.5 I/O Status Block 2–39
2.6 Disk Driver Programming Example 2–40

3 Magnetic Tape Drivers

3.1 Magnetic Tape Controllers and Drives 3–1
3.1.1 TM03 Magnetic Tape Controller (VAX Only) 3–1
3.1.2 TS11 Magnetic Tape Controller (VAX Only) 3–1
3.1.3 TM78 and TM79 Magnetic Tape Controllers (VAX Only) 3–1
3.1.4 TU80 Magnetic Tape Subsystem (VAX Only) 3–1
3.1.5 TA81 Magnetic Tape Subsystem 3–1
3.1.6 TU81 Magnetic Tape Subsystem (VAX Only) 3–2
3.1.7 TU81-Plus Magnetic Tape Subsystem (VAX Only) 3–2
3.1.8 TA90 Magnetic Tape Subsystem 3–2
3.1.9 RV20 Write-Once Optical Drive (VAX Only) 3–2
3.1.10 TK50 Cartridge Tape System (VAX Only) 3–2

3.1.11 TK70 Cartridge Tape System (VAX Only) 3–3
3.1.12 TZ30 Cartridge Tape System 3–3
3.1.13 Read and Write Compatibility Between Cartridge Tape Systems 3–3
3.2 Driver Features 3–4
3.2.1 Dual Path Tape Drives 3–5
3.2.2 Dynamic Failover and Mount Verification 3–5
3.2.3 Tape Caching 3–5
3.2.4 Master Adapters and Slave Formatters 3–5
3.2.5 Data Check 3–6
3.2.6 Error Recovery 3–6
3.2.7 Streaming Tape Systems 3–7
3.3 Magnetic Tape Driver Device Information 3–8
3.4 Magnetic Tape Function Codes 3–9
3.4.1 Read 3–13
3.4.2 Write 3–15
3.4.3 Rewind 3–16
3.4.4 Skip File 3–16
3.4.5 Skip Record 3–16
3.4.5.1 Logical End-of-Volume Detection 3–17
3.4.6 Write End-of-File 3–17
3.4.7 Rewind Offline 3–18
3.4.8 Unload 3–18
3.4.9 Sense Tape Mode 3–18
3.4.10 Set Mode 3–19
3.4.11 Data Security Erase 3–22
3.4.12 Modify 3–22
3.4.13 Pack Acknowledge 3–22
3.4.14 Available 3–22
3.4.15 Flush 3–22
3.5 I/O Status Block 3–23
3.6 Magnetic Tape Driver Programming Examples 3–23

4 Mailbox Driver

4.1 Mailbox Operations 4–1
4.1.1 Creating Mailboxes 4–1
4.1.2 Deleting Mailboxes 4–3
4.1.3 Mailbox Message Format 4–4
4.1.4 Mailbox Protection 4–4
4.2 Mailbox Driver Device Information 4–4
4.3 Mailbox Function Codes 4–5
4.3.1 Read 4–5
4.3.2 Write 4–9
4.3.3 Write End-of-File Message 4–11
4.3.4 Set Attention AST 4–11
4.3.5 Wait for Writer/Reader 4–14
4.3.6 Set Protection 4–14
4.3.7 Get Mailbox Information 4–15
4.4 I/O Status Block 4–15
4.5 Mailbox Driver Programming Examples 4–17

5 Terminal Driver

5.1 Supported Terminal Devices 5–1
5.2 Terminal Driver Features 5–2
5.2.1 Input Processing 5–3
5.2.1.1 Command Line Editing and Command Recall 5–3
5.2.1.2 Control Characters and Special Keys 5–4
5.2.1.3 Read Verify 5–7
5.2.1.4 Escape and Control Sequences 5–7
5.2.1.5 Type-Ahead Feature 5–9
5.2.1.6 Line Terminators 5–10
5.2.1.7 Special Operating Modes 5–10
5.2.2 Output Processing 5–10
5.2.2.1 Duplex Modes 5–10
5.2.2.2 Formatting of Output 5–11
5.2.2.3 SET HOST Facility and Output Buffering 5–11
5.2.3 Dial-Up Support 5–13
5.2.3.1 Modem Signal Control 5–13
5.2.3.2 Hangup on Logging Out 5–17
5.2.3.3 Preservation of a Process Across Hangups 5–17
5.2.4 Terminal/Mailbox Interaction 5–18
5.2.5 Autobaud Detection 5–19
5.2.6 Out-of-Band Control Character Handling 5–20
5.3 Terminal Driver Device Information 5–20
5.3.1 Terminal Characteristics Categories 5–26
5.4 Terminal Function Codes 5–27
5.4.1 Read 5–28
5.4.1.1 Function Modifier Codes for Read QIO Functions 5–29
5.4.1.2 Read Function Terminators 5–30
5.4.1.3 Itemlist Read Operations 5–31
5.4.1.4 Read Verify Function 5–36
5.4.2 Write 5–37
5.4.2.1 Function Modifier Codes for Write QIO Functions 5–38
5.4.2.2 Write Function Carriage Control 5–38
5.4.3 Set Mode 5–40
5.4.3.1 Hangup Function Modifier 5–44
5.4.3.2 Enable Ctrl/C AST and Enable Ctrl/Y AST Function Modifiers . . . 5–44
5.4.3.3 Set Modem Function Modifier 5–45
5.4.3.4 Loopback Function Modifier 5–47
5.4.3.5 Enable Out-of-Band AST Function Modifier 5–48
5.4.3.6 Broadcast Function Modifier 5–49
5.4.4 LAT Port Driver QIO Interface 5–49
5.4.4.1 LAT Port Types 5–50
5.4.4.2 LAT Port Driver Functions 5–51
5.4.4.3 Creating and Configuring LAT Entities 5–51
5.4.4.4 Obtaining Information About LAT Entities 5–59
5.4.4.5 Programming Application Ports 5–74
5.4.4.6 Programming Application Services and Dedicated Ports 5–76
5.4.4.7 Programming Forward Ports 5–76
5.4.4.8 Queue Change Notification (Alpha Only) 5–77
5.4.4.9 Hangup Notification 5–78
5.4.5 Sense Mode and Sense Characteristics 5–78
5.4.5.1 Type-ahead Count Function Modifier 5–79
5.4.5.2 Read Modem Function Modifier 5–80
5.4.5.3 Broadcast Function Modifier 5–81

5.5 I/O Status Block 5–81
5.6 Terminal Driver Programming Examples 5–85

6 Pseudoterminal Driver

6.1 Pseudoterminal Operations 6–1
6.1.1 Creating a Pseudoterminal 6–1
6.1.2 Canceling a Request 6–2
6.1.3 Deleting a Pseudoterminal 6–2
6.2 Pseudoterminal Driver Features 6–3
6.3 Pseudoterminal Driver Device Information 6–3
6.4 I/O Buffers 6–3
6.5 Pseudoterminal Functions 6–4
6.5.1 Reading Data 6–5
6.5.2 Writing Data 6–5
6.5.3 Using Write with Echo 6–5
6.5.4 Flow Control 6–5
6.5.5 Event Notification 6–6
6.5.5.1 Input Flow Control 6–6
6.5.5.2 Output Stop 6–6
6.5.5.3 Output Resume 6–6
6.5.5.4 Characteristics Changed 6–7
6.5.5.5 Output Abort 6–7
6.5.5.6 Terminal Driver Read Events 6–7
6.6 Pseudoterminal Driver Programming Example 6–7
6.6.1 Design Overview 6–8

7 Shadow-Set Virtual Unit Driver

7.1 Introduction 7–1
7.2 Configurations 7–2
7.2.1 Hardware Supported 7–2
7.2.2 Compatible Disk Drives and Volumes 7–2
7.3 Driver Functions 7–2
7.3.1 CRESHAD 7–3
7.3.2 ADDSHAD 7–4
7.3.3 COPYSHAD 7–5
7.3.4 REMSHAD 7–5
7.3.5 AVAILABLE 7–6
7.3.6 SENSECHAR 7–6
7.3.7 Read and Write Functions 7–7
7.4 Error Processing 7–8

8 Local Area Network (LAN) Device Drivers

8.1 Relevant Terms 8–2
8.2 LAN Characteristics 8–2
8.2.1 Driver Initialization and Operation 8–4
8.2.2 Ethernet Addresses 8–6
8.2.2.1 Format of Ethernet Addresses 8–6
8.2.2.2 Ethernet Address Classifications 8–6
8.2.2.3 Selecting an Ethernet Physical Address 8–6
8.2.2.4 Ethernet Physical and Multicast Address Values 8–7
8.2.2.5 Token Ring Functional Address Mapping (Alpha Only) 8–8

8.2.3 Industry-Standard Support 8–9
8.3 Procedures for Configuring ISA Devices 8–9
8.3.1 OpenVMS LAN Devices Requiring Configuration 8–10
8.3.1.1 DE202 Ethernet 8–10
8.3.1.2 DW110 Token Ring 8–11
8.4 Frame Formats 8–12
8.4.1 CSMA/CD Frames 8–13
8.4.2 FDDI Frames 8–14
8.4.3 Token Ring Frames (Alpha Only) 8–15
8.4.4 802.2/802.1 Headers 8–15
8.4.5 Token Ring Source Routing Header (Alpha Only) 8–16
8.5 Format Parameter 8–16
8.6 Features of Formats 8–19
8.6.1 Ethernet Packet Format 8–20
8.6.1.1 Ethernet Protocol Types 8–20
8.6.1.2 Ethernet Packet Padding 8–21
8.6.1.3 Protocol Type Sharing 8–21
8.6.2 IEEE 802 Packet Format 8–22
8.6.2.1 Class I Service Packet Format 8–22
8.6.2.2 User-Supplied Service Header Format 8–24
8.6.2.3 Service Access Point (SAP) Use and Restrictions 8–24
8.6.3 IEEE 802 Extended Packet Format 8–25
8.6.3.1 Protocol Type PID Sharing (Alpha Only) 8–25
8.7 LAN Device Information 8–26
8.8 LAN Function Codes 8–28
8.8.1 Read 8–29
8.8.2 Write 8–32
8.8.3 Set Mode and Set Characteristics 8–35
8.8.3.1 Set Controller Mode 8–35
8.8.3.2 Set Mode Parameters for Packet Formats 8–45
8.8.3.3 Set Mode Parameter Validation 8–46
8.8.3.4 Shutdown Controller 8–47
8.8.3.5 Enable Attention AST 8–47
8.8.3.6 IO$M_SET_MAC Functional Modifier to IO$M_SETMODE 8–48
8.8.3.7 IO$M_UPDATE_MAP Functional Modifier to IO$_SETMODE (Alpha Only) 8–49
8.8.3.8 IO$M_ROUTE Functional Modifier to IO$_SETMODE (Alpha Only) 8–51
8.8.4 Sense Mode and Sense Characteristics 8–52
8.8.4.1 IO$M_SENSE_MAC Functional Modifier to IO$_SENSEMODE 8–54
8.8.4.2 IO$M_SHOW_MAP Functional Modifier to IO$_SENSMODE (Alpha Only) 8–56
8.8.4.3 IO$M_SHOW_ROUTE Functional Modifier to IO$_SENSEMODE (Alpha Only) 8–56
8.9 I/O Status Block 8–58
8.10 Application Programming Notes 8–59
8.10.1 Promiscuous Mode 8–59
8.10.2 Local Area Network Programming Examples 8–59
8.11 References 8–70

A I/O Function Codes

A.1 ACP-QIO Interface Driver A–1
A.2 Disk Drivers A–2
A.3 Magnetic Tape Drivers A–3
A.4 Mailbox Driver A–5
A.5 Terminal Driver A–6
A.6 Local Area Network Device Drivers A–8

B Tables

B.1 DEC Multinational Character Set B–1
B.2 Terminal Sequences and Modes B–9

C Control Connection Routines

PTD$CANCEL C–2
PTD$CREATE C–3
PTD$DELETE C–6
PTD$READ C–7
PTD$READW C–9
PTD$SET_EVENT_NOTIFICATION C–10
PTD$WRITE C–13

Index

Examples

2–1 DISK_DRIVER.MAR Disk Driver Programming Example 2–40
3–1 Defining the P1 Parameter in a IO$_SKIPRECORD QIO 3–13
3–2 Device Characteristic Program Example 3–24
3–3 Set Mode and Sense Mode Program Example 3–24
3–4 MAGNETIC_TAPE.MAR Device Characteristic Program Example . . . 3–26
4–1 Mailbox Driver Program Example 1 4–18
4–2 Mailbox Driver Program Example 2 4–20
4–3 Mailbox Driver Program Example 3 4–22
5–1 LAT.C Terminal Driver Programming Example 5–85
5–2 FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example 5–99
5–3 READ_VERIFY.MAR Terminal Driver Programming Example 5–108
6–1 Sample Pseudocode for Pseudoterminal Driver Program 6–8
8–1 Using the 'isacfg' at Console Prompt 8–11
8–2 Using the SYS$MANAGER:ISA_CONFIG.DAT Entry 8–11
8–3 Using the 'isacfg' at Console Prompt 8–12
8–4 Using the SYS$MANAGER:ISA_CONFIG.DAT entry 8–12
8–5 LANETH.MAR Local Area Network Programming Example 8–60
8–6 LAN802.MAR Local Area Network Programming Example 8–66

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	1–4
1–6	Attribute Control Block Format	1–14
1–7	ACP-QIO Record Attributes Area	1–19
1–8	ACP–QIO Attributes Statistics Block	1–21
1–9	Quota File Transfer Block	1–36
1–10	IOSB Contents—ACP-QIO Functions	1–37
2–1	Disk Physical Address	2–6
2–2	Dual-Ported Disk Drives	2–9
2–3	Audio Control Block (AUCB)	2–21
2–4	Output Channel Selection and Volume Settings for CD–ROM Ports as Used by the SET_VOLUME Function	2–24
2–5	Starting Physical Address	2–32
2–6	Physical Cylinder Number Format	2–33
2–7	IOSB Contents	2–39
2–8	IOSB Contents for the Sense Mode Function	2–40
3–1	IO$_SKIPFILE Argument	3–16
3–2	IO$_SKIPRECORD Argument	3–17
3–3	Sense Mode P1 Buffer	3–19
3–4	Set Mode Characteristics Buffer for IO$_SETMODE	3–20
3–5	Set Mode Characteristics Buffer for IO$_SETCHAR	3–20
3–6	IOSB Contents	3–23
4–1	Multiple Mailbox Channels	4–2
4–2	$QIO READ STREAM Operation	4–7
4–3	Read Mailbox	4–8
4–4	Write Mailbox	4–10
4–5	Write Attention AST (Read Unsolicited Data)	4–12
4–6	Read Attention AST	4–13
4–7	Protection Mask	4–15
4–8	IOSB Contents—Read Function	4–16
4–9	IOSB Contents—Write Function	4–16
4–10	IOSB Contents—Set Protection Function	4–16
4–11	IOSB Contents—Get Mailbox Information Function	4–16
5–1	Modem Control: Two-Way Simultaneous Operation	5–15
5–2	Terminal Mailbox Message Format	5–19
5–3	Short and Long Forms of Terminator Mask Quadwords	5–31
5–4	Itemlist Read Descriptor	5–32
5–5	P4 Carriage Control Specifier	5–38
5–6	Write Function Carriage Control (Prefix and Postfix Coding)	5–41
5–7	Set Mode and Set Characteristics Buffers	5–42
5–8	Relationship of Out-of-Band Function with Control Characters	5–46
5–9	Set Mode P1 Block	5–47

5–10	Example SETMODE Itemlist	5–53
5–11	Sense Mode Characteristics Buffer	5–79
5–12	Sense Mode Characteristics Buffer (type-ahead)	5–80
5–13	Sense Mode P1 Block	5–80
5–14	IOSB Contents—Read Function	5–82
5–15	IOSB Contents—Itemlist Read Function	5–82
5–16	IOSB Contents—Write Function	5–83
5–17	IOSB Contents—Set Mode, Set Characteristics, Sense Mode, and Sense Characteristics Functions	5–83
5–18	IOSB Contents—LAT Port Driver Function	5–84
6–1	Buffer Layout	6–4
7–1	I/O Status Block for Copy Operations	7–5
7–2	I/O Status Block for Copy Information	7–7
8–1	Typical Ethernet Configuration	8–4
8–2	LAN Frame Formats	8–13
8–3	CSMA/CD Frame with Ethernet Header	8–13
8–4	CSMA/CD Frame with IEEE 802.3 Header	8–14
8–5	FDDI Frame Format	8–14
8–6	Token Ring Frame Format (Alpha Only)	8–15
8–7	802.2 Header	8–15
8–8	802.1 Header	8–16
8–9	802.1 Header Subfields	8–16
8–10	Source Routing Field (Alpha Only)	8–16
8–11	Frames with Ethernet Format	8–17
8–12	Frames with 802 Format	8–18
8–13	Frames with 802E Format	8–19
8–14	Class I Service 802.2 Header	8–23
8–15	User-Supplied Service 802.2 Header	8–24
8–16	DSAP and SSAP Format	8–24
8–17	802 Extended Header	8–25
8–18	DVI$_DEVDEPEND Returns	8–28
8–19	Read Function P5 Buffer	8–31
8–20	Write Function P5 Buffer	8–33
8–21	P2 Extended Characteristics Buffer	8–36
8–22	Format of IO$M_UPDATE_MAP Setmode P2 Buffer (Alpha Only)	8–50
8–23	Format of the IO$M_ROUTE P2 Buffer (Alpha Only)	8–51
8–24	Sense Mode P1 Characteristics Buffer	8–53
8–25	Sense Mode Attribute Buffer	8–54
8–26	Format of IO$M_SHOW_MAP P2 Buffer (Alpha Only)	8–56
8–27	Format of IO$M_SHOW_ROUTE P2 Buffer (Alpha Only)	8–57
8–28	IOSB Contents	8–58
C–1	Device Characteristics Buffer	C–4

Tables

1–1	Contents of the File Information Block	1–5
1–2	FIB Fields (Lookup Control)	1–7
1–3	FIB Fields (Access Control)	1–9
1–4	FIB Fields (Extend Control)	1–11
1–5	FIB Fields (Truncate Control)	1–13
1–6	Attribute Control Block Fields	1–14
1–7	ACP-QIO Attributes	1–15
1–8	File Characteristics Bits	1–18
1–9	ACP Record Attributes Values	1–19
1–10	Contents of the Statistics Block	1–22
1–11	IO$_CREATE and the File Information Block	1–23
1–12	IO$_ACCESS and the File Information Block	1–26
1–13	FIB Fields (Movefile)	1–30
1–14	IO$_ACPCONTROL and the File Information Block	1–33
1–15	Magnetic Tape Operations and the File Information Block	1–34
1–16	Disk Quota Functions (Enable/Disable)	1–35
1–17	Disk Quota Functions (Individual Entries)	1–35
2–1	SCSI Disk Class Driver Audio Commands	2–19
2–2	Contents of Audio Control Block	2–21
2–3	Status Codes Returned to the IOSB and AUCB by the SCSI Disk Class Driver	2–25
2–4	Disk Device Characteristics	2–27
2–5	Disk I/O Functions	2–29
3–1	Magnetic Tape Device-Independent Characteristics	3–8
3–2	Device-Dependent Information for Tape Devices	3–8
3–3	Extended Device Characteristics for Tape Devices	3–9
3–4	Magnetic Tape I/O Functions	3–10
3–5	Set Mode and Set Characteristics Magnetic Tape Characteristics	3–21
3–6	Extended Device Characteristics for Tape Devices	3–21
4–1	Mailbox Characteristics	4–4
5–1	Supported Terminal Devices	5–1
5–2	Terminal Control Characters	5–5
5–3	Control and Data Signals (Full Modem Mode Configuration)	5–16
5–4	Terminal Device-Independent Characteristics	5–21
5–5	Terminal Characteristics	5–21
5–6	Extended Terminal Characteristics	5–23
5–7	Read QIO Function Modifiers for the Terminal Driver	5–29
5–8	Item Codes for Itemlist Read Operations for the Terminal Driver	5–32
5–9	Write QIO Function Modifiers for the Terminal Driver	5–38
5–10	Write Function Carriage Control (Fortran: byte 0 not equal to 0)	5–39
5–11	Write Function Carriage Control (P4 byte 0 = 0)	5–40
5–12	Broadcast Requester IDs	5–49
5–13	LAT$C_ENT_NODE Item Codes	5–54
5–14	LAT$C_ENT_SERVICE Item Codes	5–57
5–15	LAT$C_ENT_LINK ItemCodes	5–58

5–16	LAT$C_ENT_PORT Item Codes	5–58
5–17	LAT$C_ENT_NODE Item Codes	5–61
5–18	Node Service Subblock Item Codes	5–63
5–19	Node Counters Item Codes for Port Counters Subblocks (Alpha Only)	5–63
5–20	Node Counters Item Codes	5–64
5–21	Protocol Error Item Codes	5–66
5–22	LAT$C_ENT_SERVICE Item Codes	5–67
5–23	Service Node Subblock Item Codes	5–67
5–24	Service Counters Subblock Item Codes	5–69
5–25	LAT$C_ENT_LINK Item Codes	5–69
5–26	Link Counters Item Codes	5–70
5–27	Link Counters Item Codes	5–71
5–28	LAT$C_ENT_PORT Item Codes	5–72
5–29	LAT SENSMODE Queue Entries (Alpha Only)	5–74
5–30	IO$M_LT_CONNECT Request Status	5–75
5–31	Byte IOSB+5 Status Information	5–83
5–32	LAT Rejection Codes	5–84
7–1	Functions of the Shadow Set Virtual Unit Driver	7–3
8–1	Supported Communication Devices	8–1
8–2	LAN Device Names and Devices	8–5
8–3	Address Mappings of Token Ring Drivers	8–8
8–4	ISA Configuring Concepts	8–10
8–5	Ethernet Controller Device Characteristics	8–26
8–6	Ethernet Controller Unit and Line Status	8–28
8–7	Error Summary Bits	8–28
8–8	LAN I/O Functions	8–28
8–9	Maximum User Data Sizes	8–32
8–10	Maximum Message Sizes	8–34
8–11	P2 Attributes	8–36
8–12	Set Mode Parameters for Packet Formats	8–45
8–13	Medium Specific Parameters of IO$M_SET_MAC	8–48
8–14	Parameters of IO$M_SENSE_MAC	8–54
8–15	State of the Entry	8–58
8–16	Rules for Promiscuous Mode Operation	8–59
B–1	DEC Multinational Character Set	B–1
B–2	Sequences and Modes	B–9
C–1	Control Connection Routines	C–1
C–2	Symbolic Names Defined by $PTDDEF Macro	C–11

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 drivers. OpenVMS users who do not require such detailed knowledge of I/O drivers can use the device-independent services described in the *OpenVMS Record Management Services Reference Manual*.

Document Structure

This manual is organized into the following chapters and appendixes.

- Chapter 1 describes the Queue I/O (QIO) interface to file system ancillary control processes (ACPs).
- Chapters 2 through 9 describe the use of file-structured and real-time I/O device drivers, the drivers for storage devices such as disks and magnetic tapes, and supported terminal devices:
  - Chapter 2 discusses the disk drivers.
  - Chapter 3 discusses the magnetic tape drivers.
  - Chapter 4 discusses the mailbox driver.
  - Chapter 5 discusses the terminal driver.
  - Chapter 6 discusses the pseudoterminal driver.
  - Chapter 7 discusses the shadow-set virtual unit driver.
  - Chapter 8 discusses the local area network (LAN) device drivers.
- Appendix A summarizes the QIO function codes, arguments, and function modifiers used by the drivers listed previously.
- Appendix B lists the DEC Multinational Character Set and the ANSI and Digital-private escape sequences for terminals.
- Appendix C describes the calling conventions for the pseudoterminal driver's control connection routines.

Device Driver Support for OpenVMS Alpha 64-bit Addressing

Alpha

The OpenVMS Alpha operating system provides support for 64-bit virtual memory addressing, which makes the 64-bit virtual address space defined by the Alpha architecture available to the OpenVMS Alpha operating system and to application programs. In the 64-bit virtual address space, both process-private and system virtual address space extend beyond 2 GB. By using 64-bit addressing features, programmers can create images that map and access data beyond the limits of 32-bit virtual addresses.

Input and output operations can be performed directly to and from the 64-bit addressable space by means of RMS services, the $QIO system service, and most of the device drivers supplied with OpenVMS Alpha systems. A device driver declares support for 64-bit addresses individually by I/O function code. Disk and tape device drivers support 64-bit addresses for data transfers to and from disk and tape devices on the virtual, logical, and physical read and write functions. For example, the OpenVMS SCSI disk class driver, SYS$DKDRIVER, supports 64-bit addresses on the IO$_READVBLK and IO$_WRITEVBLK functions, but not on the IO$_AUDIO function. The device drivers, function codes, and $QIO arguments that support 64-bit addressing are indicated in the appropriate chapters of this manual.

For more information about the OpenVMS Alpha device drivers that support 64-bit addressing, see the *OpenVMS Alpha Guide to 64-Bit Addressing*. To find out how to modify a customer-written device driver to support 64-bit addressing, see the *OpenVMS Alpha Guide to Upgrading Privileged-Code Applications*. ♦

Related Documents

The following documents provide additional information:

- *OpenVMS System Services Reference Manual*
- *OpenVMS Programming Environment Manual*
- *OpenVMS Record Management Services Reference Manual*
- *DECnet for OpenVMS Networking Manual*
- *OpenVMS VAX Device Support Manual*
- *OpenVMS Alpha Guide to 64-Bit Addressing*
- *OpenVMS Alpha Guide to Upgrading Privileged-Code Applications*

For additional information on OpenVMS products and services, access the Digital OpenVMS World Wide Web site. Use the following URL:

`http://www.openvms.digital.com`

Reader's Comments

Digital welcomes your comments on this manual.

Print or edit the online form SYS$HELP:OPENVMSDOC_COMMENTS.TXT and send us your comments by:

Internet	**openvmsdoc@zko.mts.dec.com**
Fax	603 881-0120, Attention: OpenVMS Documentation, ZK03-4/U08
Mail	OpenVMS Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

How To Order Additional Documentation

Use the following table to order additional documentation or information. If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Location	Call	Fax	Write
U.S.A.	DECdirect 800–DIGITAL 800–344–4825	Fax: 800–234–2298	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Puerto Rico	809–781–0505	Fax: 809–749–8300	Digital Equipment Caribbean, Inc. 3 Digital Plaza, 1st Street, Suite 200 P.O. Box 11038 Metro Office Park San Juan, Puerto Rico 00910–2138
Canada	800–267–6215	Fax: 613–592–1946	Digital Equipment of Canada, Ltd. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International	—	—	Local Digital subsidiary or approved distributor
Internal Orders	DTN: 264–4446 603–884–4446	Fax: 603–884–3960	U.S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063–1260

ZK–7654A–GE

Conventions

The name of the OpenVMS AXP operating system has been changed to OpenVMS Alpha. Any references to OpenVMS AXP or AXP are synonymous with OpenVMS Alpha or Alpha.

The following conventions are used to identify information specific to OpenVMS Alpha or to OpenVMS VAX:

Alpha	The Alpha icon denotes the beginning of information specific to OpenVMS Alpha.
VAX	The VAX icon denotes the beginning of information specific to OpenVMS VAX.
♦	The diamond symbol denotes the end of a section of information specific to OpenVMS Alpha or to OpenVMS VAX.

In this manual, every use of DECwindows and DECwindows Motif refers to DECwindows Motif for OpenVMS software.

The following conventions are also used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x* or GOLD *x*	A sequence such as PF1 *x* or GOLD *x* indicates that you must first press and release the key labeled PF1 or GOLD and then press and release another key or a pointing device button. GOLD key sequences can also have a slash (/), dash (–), or underscore (_) as a delimiter in EVE commands.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	Horizontal ellipsis points in examples indicate one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	Vertical ellipsis points indicate the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In command format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification or in the syntax of a substring specification in an assignment statement.)
{ }	In command format descriptions, braces indicate a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in Bookreader versions of the manual.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five alphanumeric characters).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`Monospace type`	Monospace type indicates code examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.

-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

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 OpenVMS RMS (Record Management Services) and is transparent to the user process. However, user processes can request ACP functions directly by issuing a Queue I/O (QIO) request and specifying an ACP function code, as shown in Figure 1–1.

Executing physical and logical input/output (I/O) operations on a device that is managed by a file ACP interferes with the operation of the ACP and can result in such unpredictable consequences as system failure.

In addition to the ACP, the XQP (extended QIO processor) facility supplements the QIO driver's functions when performing virtual I/O operations on file-structured devices. However, rather than being a separate process, the XQP executes as a kernel-mode thread in the process of its caller.

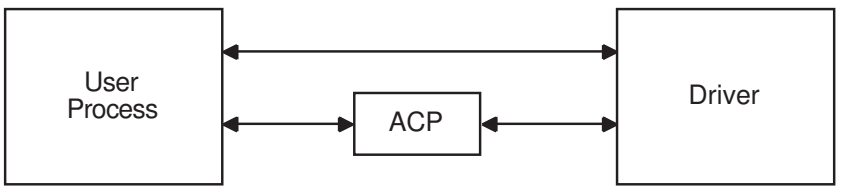
An XQP is provided to support Files–11 ODS-2 (On-Disk Structure Level 2) disks as the base file system, and an ACP is provided for ANSI standard X3.27 magnetic tapes.

VAX

On VAX systems, an ACP is provided for supporting Files–11 ODS-1 (On-Disk Structure Level 1) disks.♦

This chapter describes the QIO interface to ACPs for disk and magnetic tape devices (file system ACPs). The sample program in Chapter 3 performs QIO operations to the magnetic tape ACP.

Figure 1–1 ACP–QIO Interface



ZK–0635–GE

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

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

$IODEF
$FIBDEF
$ATRDEF
$SBKDEF

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

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

- IO$_CREATE—Creates a directory entry or file
- IO$_ACCESS—Searches a directory for a specified file and accesses 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:

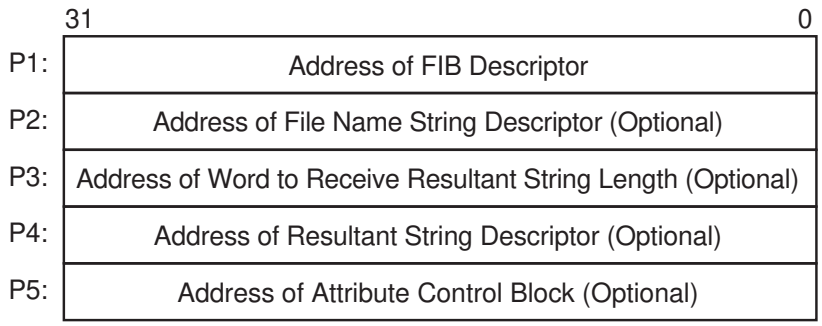
- IO$M_ACCESS—Opens a file on the user's channel
- IO$M_CREATE—Creates a file
- IO$M_DELETE—Deletes a file or marks it for deletion
- IO$M_DMOUNT—Dismounts a volume

In addition to the function codes and modifiers, 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. Argument P4 is the address of a descriptor for a buffer to receive the resultant file name string. Both of these arguments are optional.

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



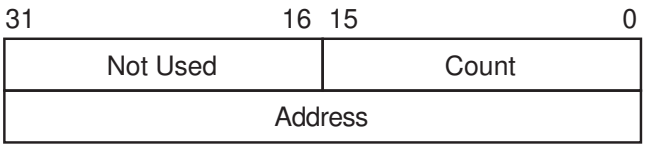
ZK–0636–GE

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

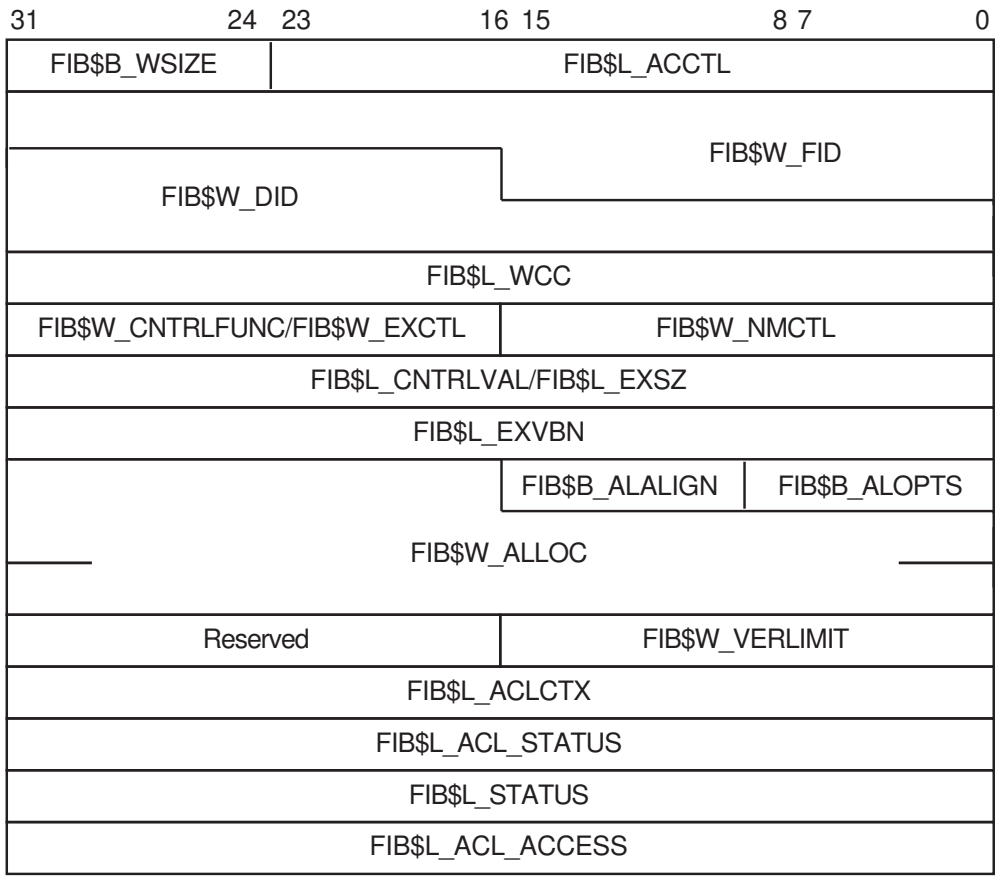


ZK–0637–GE

1.2 File Information Block (FIB)

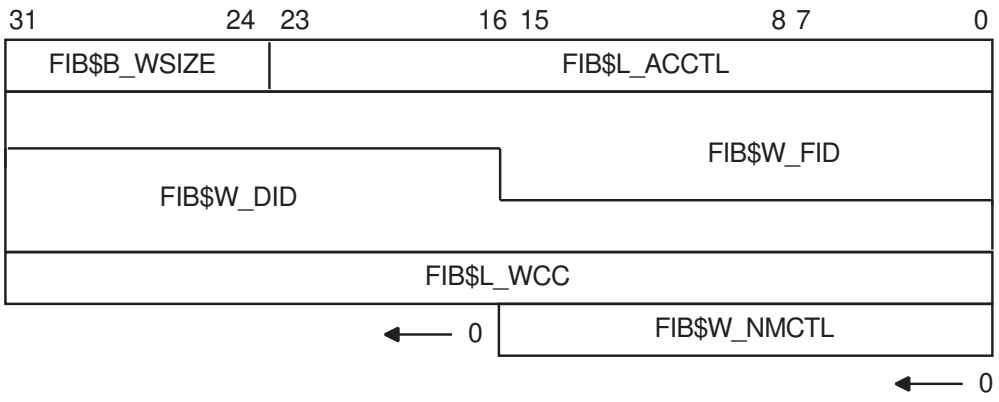
The file information block (FIB) contains much of the information that is exchanged between the user process and the ACP. Figure 1–4 shows the format of the FIB. The FIB must be writable. Because the FIB is passed by a descriptor (see Figure 1–3), its length can vary. Thus, a short FIB can be used in 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.

Figure 1–4 File Information Block Format



ZK–0638–GE

Figure 1–5 Typical Short File Information Block



ZK–0639–GE

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.
		For Files–11 On-Disk Structure Level 1 and Level 2, the following subfields are defined.
	FIB$W_FID_NUM	File number.
	FIB$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.
		For Files–11 On-Disk Structure Level 1 and Level 2, 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 1.6.8). This field overlays FIB$W_EXCTL.
FIB$L_EXSZ		Specifies the number of blocks to be allocated in an extend operation on a disk file.

(continued on next page)

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

Field	Subfields	Meaning
FIB$L_CNTRLVAL		Contains a control function value used in an IO$_ ACPCONTROL function (see Section 1.6.8). The interpretation of the value depends on the control function specified in FIB$W_CNTRLFUNC. This field overlays FIB$L_EXSZ.
FIB$L_EXVBN		Specifies the starting disk file virtual block number at which a file is to be truncated.
FIB$B_ALOPTS		Contains option bits that control the placement of allocated blocks. Section 1.3.3.1 describes the FIB$B_ALOPTS field flag bits.
FIB$B_ALALIGN		Contains the interpretation mode of the allocation (FIB$W_ALLOC) field.
FIB$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$B_LOC_RVN	Related file relative volume number (RVN) or placement RVN.
	FIB$B_LOC_NMX	Related file number extension.
	FIB$L_LOC_ADDR	Placement logical block number (LBN), cylinder, or virtual block number (VBN).
FIB$W_VERLIMIT		Contains the version limit of the directory entry.
FIB$L_ACLCTX		Maintains position context when processing ACL attributes from the attribute (P5) list.
FIB$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$_NORMAL is returned here.
FIB$L_STATUS		Access status. Applies to all major functions. The following bits are supported.
	FIB$V_ALT_REQ	Set to indicate whether the alternate access bit is required for the current operation. If not set, the alternate access bit is optional.
	FIB$V_ALT_GRANTED	If FIB$V_ALT_REQ = 0, the FIB bit returned from the file system is set if the alternate access check succeeded.
FIB$L_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/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$W_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	Subfields	Meaning
FIB$W_NMCTL		Name string control. The following name control bits are applicable to a lookup operation:
	FIB$V_WILD	Set if name string contains wildcards. Setting this bit causes wildcard context to be returned in FIB$L_ WCC.
	FIB$V_ALLNAM	Set to match all name field values.
	FIB$V_ALLTYP	Set to match all field type values.
	FIB$V_ALLVER	Set to match all version field values.
	FIB$V_FINDFID	Set to search a directory for the file identifier in FIB$W_FID.
FIB$W_FID		File identification. The file ID of the file found is returned in this field.
FIB$W_DID		Contains the file identifier of the directory file. This field must be a nonzero number.

(continued on next page)

Table 1–2 (Cont.) FIB Fields (Lookup Control)

Field	Subfields	Meaning
FIB$L_WCC		Maintains position context when processing wildcard directory operations.
FIB$L_ACCTL		The following access control flag is applicable to a lookup subfunction:
	FIB$V_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 (see Figure 1–2) 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
```

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 FIBV_ALLNAM, FIBV_ALLTYP, and FIB$V_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 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$V_WILD. (The name control bits FIBV_ALLNAM, FIBV_ALLTYP, and FIB$V_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$V_FINDFID. When this bit is set, the system searches the directory for an entry containing the file ID specified in FIB$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 directory file is protected against delete access, then the file name has the same protection. Consequently, a nonprivileged user 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	Subfields	Meaning
FIB$L_ACCTL		Specifies field values that control access to the file. The following access control bits are applicable to the access subfunction:
	FIB$V_WRITE	Set for write access; clear for read-only access.
	FIB$V_NOREAD	Set to deny read access to others. (You must have write privilege to the file to use this option.)
	FIB$V_NOWRITE	Set to deny write access to others.
	FIB$V_NOTRUNC	Set to prevent the file from being truncated; clear to allow truncation.
	FIB$V_DLOCK	Set to enable deaccess lock (close check). Used only for disk devices.

(continued on next page)

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

Field	Subfields	Meaning
		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.
	FIB$V_UPDATE	Set to position at start of a magnetic tape file when opening file for write; clear to position at end-of-file.
	FIB$V_READCK	Set to enable read checking of the file. Virtual reads to the file are performed using a data check operation.
	FIB$V_WRITECK	Set to enable write checking of the file. Virtual writes to the file are performed using a data check operation.
	FIB$V_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$V_NOLOCK	Set to override exclusive access to the file, allowing you to access the file when another user has the file open with FIB$V_NOREAD specified. You must have SYSPRV privilege to use this option. FIB$V_ NOREAD and FIB$V_NOWRITE must be clear for this option to work.
	FIB$V_NORECORD	Set to inhibit recording of the file's expiration date. If not set, the file's expiration date can be modified, depending on the file retention parameters of the volume.
	FIB$V_SEQONLY	Set to inform the file system that the file is to be processed sequentially only.
FIB$B_WSIZE		Controls the size of the file window used to map a disk file. The ACP uses the volume default if FIB$B_ WSIZE is 0. A value of 1 to 127 indicates the number of retrieval pointers to be allocated to the window. A value of –1 indicates that the window should be as large as necessary to map the entire file. Note that the window is charged to the user's BYTELIM quota.
FIB$W_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$V_EXTEND is set in the extend control word FIB$W_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	Subfields	Meaning
FIB$W_EXCTL		Extend control flags. The following flags are applicable to the extend subfunction:
	FIB$V_EXTEND	Set to enable extension.
	FIB$V_NOHDREXT	Set to inhibit generation of extension file headers.
	FIB$V_ALCON	Allocates contiguous space. The extend operation fails if the necessary contiguous space is not available.
	FIB$V_ALCONB	Allocates the maximum amount of contiguous space.
		If both FIB$V_ALCON and FIB$V_ALCONB are set, a single contiguous area, whose size is the largest available but not greater than the size requested, is allocated.
	FIB$V_FILCON	Marks the file contiguous. This bit can only be set if the file does not have space already allocated to it.
	FIB$V_ALDEF	Allocates the extend size (FIB$L_EXSZ) or the system default, whichever is greater.
FIB$L_EXSZ		Specifies the number of blocks to allocate to the file.
		The number of blocks actually allocated for this operation is returned in this longword. More blocks than requested can be allocated to meet cluster boundaries.
FIB$L_EXVBN		Returns the starting virtual block number of the blocks allocated. FIB$L_EXVBN must initially contain 0 blocks.
FIB$B_ALOPTS		Contains option bits that control the placement of allocated blocks. The following bits are defined:
	FIB$V_EXACT	Set to require exact placement; clear to specify approximate placement. If this bit is set and the specified blocks are not available, the extend operation fails.
	FIB$V_ONCYL	Set to locate allocated space within a cylinder. This option functions correctly only when FIB$V_ALCON or FIB$V_ALCONB is specified.
FIB$B_ALALIGN		Contains the interpretation mode of the allocation (FIB$W_ALLOC) field. One of the following values can be specified:
	(zero)	No placement data. The remainder of the allocation field is ignored.
	FIB$C_CYL	Location is specified as a byte relative volume number (RVN) in FIB$B_LOC_RVN and a cylinder number in FIB$L_LOC_ADDR.
	FIB$C_LBN	Location is specified as a byte RVN in FIB$B_LOC_ RVN, followed by a longword logical block number (LBN) in FIB$L_LOC_ADDR.

(continued on next page)

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

Field	Subfields	Meaning
	FIB$C_VBN	Location is specified as a longword virtual block number (VBN) of the file being extended in FIB$L_LOC_ADDR. A 0 VBN or one that fails to map indicates the end of the file.
	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$V_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$V_TRUNCATE is set in the extend control word FIB$W_EXCTL.

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	Subfields	Meaning
FIB$W_EXCTL		Extend control flags. The following flags are applicable to the truncate subfunction:
	FIB$V_TRUNC	Must be set to enable truncation.
	FIB$V_MARKBAD	Set to append the truncated blocks to the bad block file, instead of returning them to the free storage pool. Only one cluster can be deallocated. This is most easily accomplished by specifying the last VBN of the file in FIB$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$L_EXVBN. Because of cluster round-up, this value might be greater than the value specified. If FIB$V_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.

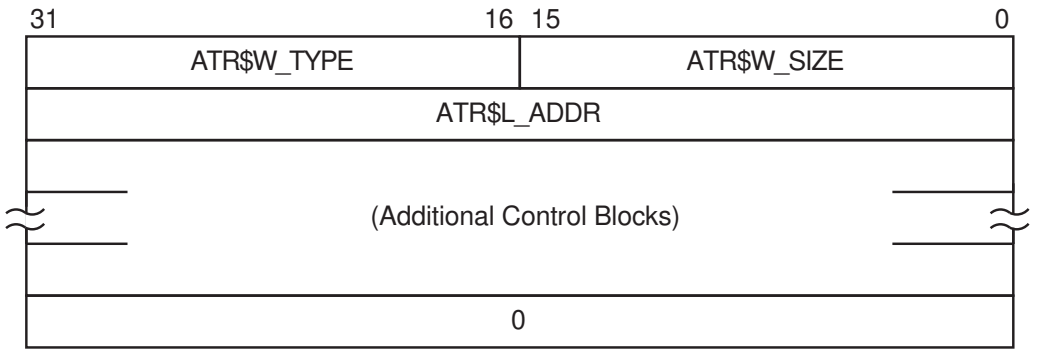
1.3.5 Read/Write Attributes

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–0640–GE

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$S_FILNAM) uses only 6 bytes, but it is always accompanied by the file type and file version, so a total of 10 bytes is required. Each attribute has two names: one for the code (for example, ATR$C_UCHAR) and one for the size (for example, ATR$S_UCHAR).

Table 1–7 ACP-QIO Attributes

Attribute Name[6]	Maximum Size (bytes)	Meaning
ATR$C_UCHAR[2] [3]	4	4-byte file characteristics. (The file characteristics bits are listed following this table.)
ATR$C_RECATTR[4]	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$C_EXPDAT[2]	7	Expiration date in ASCII. Format: DDMMMYY.
ATR$C_STATBLK	32	Statistics block. This attribute is read only. Section 1.5 describes the statistics block in detail.
ATR$C_HEADER	512	Complete file header. This attribute is read only.
ATR$C_BLOCKSIZE	2	Magnetic tape block size.
ATR$C_USERLABEL	80	User file label. This attribute is not supported for disk devices.
ATR$C_ASCDATES[2] [3]	35	Revision count (2 binary bytes), revision date, creation date, and expiration date, in ASCII. Format: DDMMMYY (revision date), HHMMSS (time), DDMMMYY (creation date), HHMMSS (time), DDMMMYY (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.

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

[3]Not supported on write operations to MTAACP; defaults are returned on read operations.

[4]Locked (cannot be written to while the file is locked).

[6]Attributes 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.

(continued on next page)

Table 1–7 (Cont.) ACP-QIO Attributes

Attribute Name[6]	Maximum Size (bytes)	Meaning
ATR$C_ASCNAME	86	Disk: file name, type, and version, in ASCII, including punctuation. Format: name.type;version. Magnetic tape: contains 17-character file identifier (ANSI a); no version number. Overrides all other file name and file type specifications if supplied on input operations. If specified on an access operation and you want only a value to be returned, specify (in ATR$W_SIZE) a buffer of greater than 17 bytes.
ATR$C_CREDATE[2]	8	64-bit creation date and time.
ATR$C_REVDATE[2] [4]	8	64-bit revision date and time.
ATR$C_EXPDATE[2]	8	64-bit expiration date and time.
ATR$C_BAKDATE[4] [5] [8] [10]	8	64-bit backup date and time.
ATR$C_UIC[2]	4	4-byte file owner UIC.
ATR$C_FPRO[2] [4]	2	File protection.
ATR$C_RPRO [10]	2	2-byte record protection.
ATR$C_ACLEVEL[2] [4] [5] [10]	1	File access level.
ATR$C_SEMASK [10]	8	File security mask and limit.
ATR$C_UIC_RO	4	4-byte file owner UIC. This attribute is read only.
ATR$C_DIRSEQ [10]	2	Directory update sequence count.
ATR$C_BACKLINK [10]	6	File back link pointer.
ATR$C_JOURNAL [10]	1	Journal control flags.
ATR$C_HDR1_ACC	1	ANSI magnetic tape header label accessibility character.
ATR$C_ADDACLENT [7] [10] [11]	255	Adds an ACE to the beginning of the ACL when the ACE context value is 0; to the end of the ACL when the ACE context value is –1; or at a location pointed to by a prior ACL$C_FNDACETYP or ACL$C_FNDACLENT.

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

[4]Locked (cannot be written to while the file is locked).

[5]For Files–11 C/D, returns 0.

[6]Attributes 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.

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

[8]Can be written only by the system, owner, or someone holding READALL privilege.

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

[11]The status from this attribute operation is returned in FIB$L_ACL_STATUS.

(continued on next page)

Table 1–7 (Cont.) ACP-QIO Attributes

Attribute Name[6]	Maximum Size (bytes)	Meaning
ATR$C_DELACLENT [7] [10] [11]	255	Deletes an access control entry pointed to by the buffer address or, if the buffer address is 0, the ACE pointed to by a prior ACL$C_FNDACETYP or ACL$C_ FNDACLENT.
ATR$C_MODACLENT [7] [10] [11]	255	Replaces the ACE pointed to by a prior ACL$C_FNDACETYP or ACL$C_FNDACLENT with the ACE pointed to by its buffer address.
ATR$C_FNDACLENT [10] [11]	255	Locates an ACE pointed to by its buffer address.
ATR$C_FNDACETYP [10] [11]	255	Locates an ACE of the type pointed to by its buffer address.
ATR$C_DELETEACL [7] [10] [11]	255	Deletes the entire ACL with the exception of protected ACEs.
ATR$C_READACL [10] [11]	512	Reads the entire ACL or as much as will fit in the supplied buffer. Only complete ACEs are transferred.
ATR$C_ACLLENGTH [10] [11]	4	Returns the size, in bytes, of the object's ACL.
ATR$C_READACE [10] [11]	255	Reads the ACE pointed to by ACL$C_FNDACETYP or ACL$C_ FNDACLENT into the buffer.
ATR$C_RESERVED [9] [10]	380	Modifies reserve area.
ATR$C_HIGHWATER [10]	4	High-water mark (user read-only).
ATR$C_PRIVS_USED[10]	4	Privileges used to gain access. This attribute can only be retrieved on the initial file access or create operation.
ATR$C_MATCHING_ACE [8] [10]	255	ACE used to gain access (if any). This attribute can only be retrieved on the initial file access or create operation.
ATR$C_ACCESS_MODE	1	Access mode for following attribute descriptors.
ATR$C_FILE_SPEC[10]	512	Convert FID to file specification.
ATR$C_BUFFER_OFFSET [3]	2	Offset length for ANSI magnetic tape header label buffer.

[3]Not supported on write operations to MTAACP; defaults are returned on read operations.

[6]Attributes 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.

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

[8]Can be written only by the system, owner, or someone holding READALL privilege.

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

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

[11]The status from this attribute operation is returned in FIB$L_ACL_STATUS.

(continued on next page)

Table 1–7 (Cont.) ACP-QIO Attributes

Attribute Name[6]	Maximum Size (bytes)	Meaning
ATR$C_DELETE_ALL [7 10 11]	255	Delete the entire access control list (ACL), including protected entries.
ATR$C_GRANT_ACE [10 11]	255	Return an ACE that grants or denies access to the object.
ATR$C_NEXT_ACE [10 11]	4	Advance to the next ACE in the ACL.

[6]Attributes 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.

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

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

[11]The status from this attribute operation is returned in FIB$L_ACL_STATUS.

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

Table 1–8 File Characteristics Bits

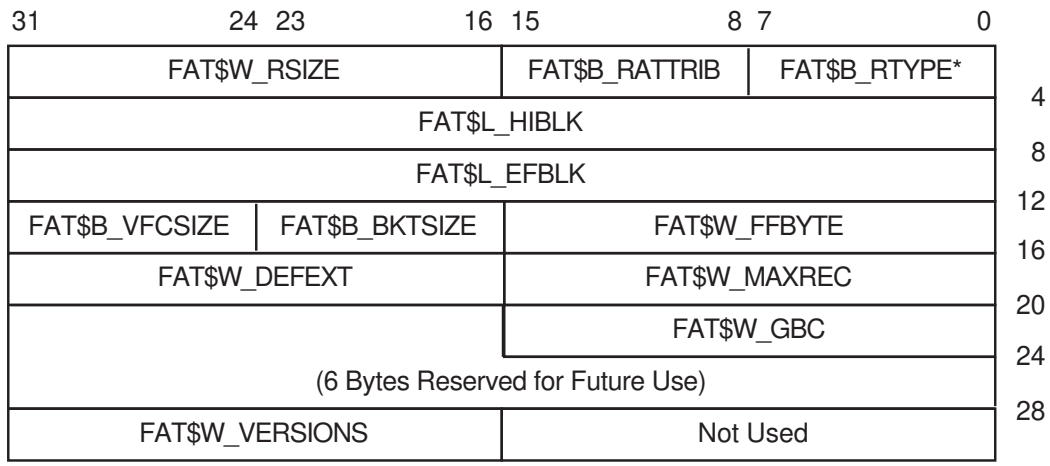
FCH$M_NOBACKUP	Do not back up file.
FCH$M_READCHECK	Verify all read operations.
FCH$M_WRITCHECK	Verify all write operations.
FCH$M_CONTIGB	Keep file as contiguous as possible.
FCH$M_LOCKED	File is deaccess-locked.
FCH$M_CONTIG	File is contiguous.
FCH$M_BADACL	File's ACL is corrupt.
FCH$M_SPOOL	File is 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.
FCH$M_ASSOCIATED[1]	File has an associated file.
FCH$M_EXISTENCE[1]	Suppress existence of file.
FCH$M_NOMOVE	Disable movefile operations on this file.
FCH$M_NOSHELVABLE	File is not shelvable.
FCH$M_SHELVED	File is shelved.

[1]Files–11 C/D only.

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

ZK–0641–GE

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

Table 1–9 ACP Record Attributes Values

Field Value	Meaning	
FAT$B_RTYPE	Record type. Contains FAT$V_RTYPE and FAT$V_FILEORG.	
FAT$V_RTYPE	Record type. The following bit values are defined:	
	FAT$C_FIXED	Fixed-length record
	FAT$C_VARIABLE	Variable-length record
	FAT$C_VFC	Variable-length record with fixed control
	FAT$C_UNDEFINED	Undefined record format (stream binary)
	FAT$C_STREAM	RMS stream format
	FAT$C_STREAMLF	Stream terminated by LF
	FAT$C_STREAMCR	Stream terminated by CR
FAT$V_FILEORG	File organization. The following bit values are defined:	
	FAT$C_DIRECT	Direct file organization[1]
	FAT$C_INDEXED	Indexed file organization
	FAT$C_RELATIVE	Relative file organization
	FAT$C_SEQUENTIAL	Sequential file organization

[1]Defined but not implemented.

(continued on next page)

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

Field Value	Meaning	
FAT$B_RATTRIB	Record attributes. The following bit values are defined:	
	FAT$M_FORTRANCC	FORTRAN carriage control
	FAT$M_IMPLIEDCC	Implied carriage control
	FAT$M_PRINTCC	Print file carriage control
	FAT$M_NOSPAN	No spanned records
	FAT$M_MSBRCW[4]	Record count word (RCW) is MSB formatted
FAT$W_RSIZE	Record size in bytes.	
FAT$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$W_HIBLKH	High-order 16 bits
	FAT$W_HIBLKL	Low-order 16 bits
FAT$L_EFBLK[2 3]	End-of-file VBN	
	FAT$W_EFBLKH	High-order 16 bits
	FAT$W_EFBLKL	Low-order 16 bits
FAT$W_FFBYTE[3]	First free byte in FAT$L_EFBLK.	
FAT$B_BKTSIZE	Bucket size in blocks.	
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.	

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

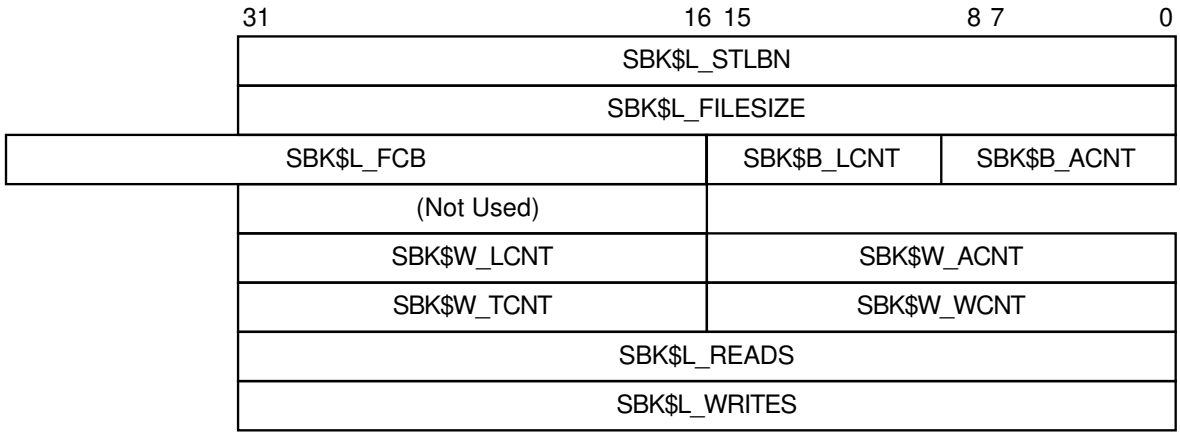
[3]When the end-of-file position corresponds to a block boundary, by convention FAT$L_EFBLK contains the end-of-file VBN plus 1, and FAT$W_FFBYTE contains 0.

[4]Variable length record format (FAT$C_VARIABLE) only.

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–0642–GE

Table 1–10 Contents of the Statistics Block

Field	Subfields	Meaning
SBK$L_STLBN		Contains the starting LBN of the file if the file is contiguous. If the file is not contiguous, this field contains a value of 0. The LBN appears as an inverted longword (the high- and low-order 16 bits are transposed for PDP–11 compatibility). The following subfields are defined:
	SBK$W_STLBNH	Starting LBN (high-order 16 bits).
	SBK$W_STLBNL	Starting LBN (low-order 16 bits).
SBK$L_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$W_FILESIZH	File size (high-order 16 bits).
	SBK$W_FILESIZL	File size (low-order 16 bits).
SBK$B_ACNT[1]		Access count (low byte). Field is for PDP–11 compatibility.
SBK$B_LCNT[1]		Lock count (low byte). Field is for PDP–11 compatibility.
SBK$L_FCB		System pool address of the file's file control block.
SBK$W_ACNT[1]		Access count (number of channels with file open currently).
SBK$W_LCNT[1]		Lock count (the number of access operations that have locked the file against writers).
SBK$W_WCNT[1]		Writer count (the number of channels that currently have the file open for write).
SBK$W_TCNT[1]		Truncate lock count (the number of access operations that have locked the file against truncation).
SBK$L_READS		Number of read operations executed for file on this channel.
SBK$L_WRITES		Number of write operations executed for file on this channel.

[1]Accesses 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).

Table 1–11 lists fields in the FIB that are applicable to the IO$_CREATE operation.

Table 1–11 IO$_CREATE and the File Information Block

Field	Subfields	Meaning
FIB$L_ACCTL		Specifies field values that control access to the file. The following bits are applicable to the IO$_CREATE function:
	FIB$V_REWIND	Set to rewind magnetic tape before creating the file. Any data currently on the tape is overwritten.
	FIB$V_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$V_REWIND nor FIB$V_CURPOS is set.) If the tape is not positioned at the end of a file, FIB$V_CURPOS creates a file at the next file position. Any data currently on the tape past the current file position is overwritten.
	FIB$V_ 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$W_ CNTRLFUNC		Specifies the following value, which allows you to control actions subsequent to EOT detection on a magnetic tape file.
FIB$W_FID		Contains the file ID of the file created or entered.
FIB$W_DID		Contains the file identifier of the directory file.

(continued on next page)

Table 1–11 (Cont.) IO$_CREATE and the File Information Block

Field	Subfields	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$V_NEWVER	Set to create file of same name with next higher version number. Only for disk devices.
	FIB$V_ SUPERSEDE	Set to supersede an existing file of the same name, type, and version. Only for disk devices.
	FIB$V_LOWVER	Set on return if a lower numbered version of the file exists. Only for disk devices.
	FIB$V_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$M_CREATE is specified, a file is created. The file ID of the file created is returned in FIB$W_FID. If the modifier IO$M_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).
- If the version number in the specified file name is a nonzero number and FIB$V_NEWVER is set, the directory entry created gets a version number one greater than the highest previously existing version of that file, or the specified version number, whichever is greater.
- If the version number in the specified file name is a nonzero number and the directory already contains a file of the same name, type, and version, the previously existing file is set aside for deletion if FIB$V_SUPERSEDE is specified. If FIB$V_SUPERSEDE is not specified, the create operation fails with a SS$_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$M_ACCESS is specified, the file is opened (see Section 1.3.2).

If the extend enable bit FIB$V_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$M_DELETE is specified), that file is deleted. If the file is deleted because the FIB$V_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 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$M_CREATE modifier is specified. The magnetic tape is positioned as specified by FIB$V_REWIND and FIB$V_CURPOS, and the file is created. The name specified by the P2 parameter is written into the file header label.

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).

Table 1–12 lists FIB fields that are applicable to the IO$_ACCESS operation.

Table 1–12 IO$_ACCESS and the File Information Block

Field	Subfields	Meaning
FIB$W_CNTRLFUNC		Specifies the value that allows the user to control actions subsequent to EOT detection on a magnetic tape file.
FIB$W_VERLIMIT		Receives the version limit for the file. Applicable only if FIB$W_DID is a nonzero number (if a directory lookup is done). Used only for disk devices.
FIB$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$_NORMAL is returned here. (For Files–11 C/D, this field is always set to SS$_NORMAL.)
FIB$L_STATUS		Alternate access status. The following bits are supported:
	FIB$V_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$V_ALT_GRANTED	If FIB$V_ALT_REQ = 0 and the alternate access check succeeded, the FIB bit returned from the file system is set.
FIB$L_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).

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

Field	Meaning
FIB$W_FID	File identification of the file being deaccessed. This field can contain a value of 0. If it does not, it must match the file identifier of the accessed file.
FIB$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$_NORMAL is returned here. (For Files–11 C/D, this field is always set to SS$_NORMAL.)

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$V_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$V_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; that is, the function returns success, but no action is performed.

The following is the function code:

- IO$_MODIFY

The following is the function modifier:

- IO$M_MOVEFILE

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).

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

Field	Subfields	Meaning
FIB$L_ACCTL		Specifies field values that control access to the file. The following bit is applicable to the IO$_MODIFY function:
	FIB$V_WRITETHRU	Specifies that the file header is to be written back to the disk. If not specified and the file is currently open, writing of the file header can be deferred to some later time.
FIB$W_VERLIMIT		If a nonzero number, specifies the version limit for 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.

1.6.4.2 Operation

If a nonzero directory ID is specified in FIB$W_DID, a lookup subfunction is executed (see Section 1.3.1). If a nonzero version limit is specified in FIB$W_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$V_EXTEND is specified in the FIB, an extend subfunction is performed (see Section 1.3.3). If FIB$V_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$M_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).
- 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	Subfields	Meaning
FIB$L_ACCTL		Specifies field values that control access to the file. The following bit is applicable to the IO$_DELETE function.
	FIB$V_WRITETHRU	Specifies that the file header is to be written back to the disk. If not specified and the file is currently open, writing of the file header can be deferred to some later time.
FIB$W_FID		Specifies the file identification to be deleted.

1.6.5.1 Operation

If a nonzero directory ID is specified in FIB$W_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$M_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 Movefile Subfunction

The movefile subfunction permits you to move the contents of a file, or part of the contents of a file, to a new disk location. This subfunction could, for example, form the basis of a disk defragmentation application.

You can disable movefile operations on specific user files by specifying the /NOMOVE qualifier on the SET FILE command. Use the DIRECTORY/FULL and the DUMP/HEADER commands to find out if movefile operations are disabled on a file.

1.6.6.1 Calling the Movefile Subfunction

A program can invoke a movefile subfunction by issuing a QIO request using the function code IO$_MODIFY and the function modifier IO$M_MOVEFILE. This section describes the various input parameters that control the processing of movefile operations together with an operational description.

1.6.6.2 Input Parameters

Table 1–13 lists the FIB fields that control the processing of a movefile subfunction.

Table 1–13 FIB Fields (Movefile)

Field	Subfields	Meaning
FIB$L_ACCTL		Movefile control flag. The following flag is applicable:
	FIB$V_NOVERIFY	Inhibits comparison of the moved blocks. If this flag is clear, the movefile operation verifies that the operation was carried out correctly by comparing the moved blocks to the original blocks.
FIB$W_FID		Specifies the file identification of the file to be moved.
FIB$W_EXCTL		Movefile control flags. The following flags apply to the movefile operation. All other FIB$W_EXCTL flags must be clear.
	FIB$V_ALCON	Specifies that the movefile operation must allocate contiguous disk space to the moved blocks. If the necessary contiguous space is not available, the movefile operation fails. The movefile operation sets this flag if the file was previously marked contiguous.
	FIB$V_ALCONB	Specifies that the movefile operation should attempt to allocate contiguous disk space to the moved blocks. That is, if the movefile operation cannot allocate contiguous space to all the moved blocks, it allocates contiguous space to as many of the blocks as possible. The movefile operation sets this flag if the file was previously marked contiguous best try.
	FIB$V_FILCON	Specifies that the entire file must be made contiguous. Do not set this flag without also setting the FIB$V_ALCON flag. If the FIB$V_FILCON flag is set, and either the FIB$V_ALCON flag is clear or the file would not be made contiguous by moving the specified virtual blocks, the movefile operation fails. The movefile operation sets this flag if the file was previously marked contiguous.
	FIB$V_NOPLACE	Specifies that placement information will not be recorded in the file header. If this flag is clear and you specify exact placement for the moved blocks, placement information for those blocks will be recorded in the file header. If this flag is set, the placement information will not be recorded. You specify exact placement through the FIBV_EXACT, FIBC_LBN, and FIB$L_LOC_ADDR fields.

(continued on next page)

Table 1–13 (Cont.) FIB Fields (Movefile)

Field	Subfields	Meaning
FIB$B_ALOPTS		Flags that control the placement of the allocated blocks. Currently, only the FIB$V_EXACT flag applies to the movefile operation. All other FIB$B_ ALOPTS flags must be clear. The following flag is applicable:
	FIB$V_EXACT	Set to require exact placement. If this flag is set and the specified blocks are not available, the movefile operation fails.
FIB$B_ALALIGN		Contains the interpretation mode of the allocation field (FIB$W_ALLOC). You can specify a field value of zero or you can specify the symbolic value FIB$C_ LBN. If you specify zero, the allocation field is ignored.
FIB$W_ALLOC		Contains the desired location of the blocks being allocated. Interpretation of the field is controlled by the FIB$B_ALALIGN field. The following subfields are defined:
	FIB$B_LOC_RVN	Specifies the relative volume number (RVN) of the volume to which the blocks are moved. Do not specify a value for this field unless you have set the FIB$V_CHANGE_VOL flag.
	FIB$L_LOC_ADDR	If the FIB$C_LBN and FIB$V_EXACT flags are set, specifies the starting logical address to which the blocks are moved.
FIB$L_MOV_SVBN		Specifies the virtual block number (VBN) of the first block to be moved. The starting virtual block number must correspond to the first block of a disk cluster. The value must be greater than zero and it must not exceed the number of virtual blocks allocated to the file. If you specify an invalid value, the movefile operation fails.
FIB$L_MOV_VBNCNT		Specifies the number of consecutive virtual blocks to be moved. This value must be a multiple of the disk cluster size and it must not exceed the difference between the greatest VBN allocated to the file and the FIB$L_MOV_SVBN value. If you specify a value of zero, the movefile operation moves all the virtual blocks between the FIB$L_MOV_SVBN value and the greatest VBN. If you specify an invalid value, the movefile operation fails.

1.6.6.3 Operation

A program can perform a movefile operation on a file if the following conditions are met:

- The program has write and control access to the file.
- The file is closed.
- Movefile operations are not disabled on the file.

Movefile operations are automatically disabled on critical system files. You can disable movefile operations on specific user files by specifying the /NOMOVE qualifier with the SET FILE command.

- The operation is not interrupted.

  If the movefile operation is interrupted by any other operation, such as a read or write operation, the movefile operation aborts and the file remains in its original position.

The movefile operation moves a specified number of consecutive virtual blocks to new logical blocks on disk, beginning with the virtual block specified in the FIB$L_SVBN field.

The number of blocks moved is specified in the FIB$L_VBNCNT field. To move an entire file, specify FIB$L_VBNCNT as 0 and FIB$L_SVBN as 1.

To specify a starting logical block for the moved blocks, specify the logical block address in the FIB$L_LOC_ADDR subfield and set the FIB$C_LBN and the FIB$V_EXACT flags.

To move the blocks to another volume, or move blocks that span more than one volume, set the FIB$V_CHANGE_VOL flag of the FIB$L_ACCTL field. Use the FIB$B_LOC_RVN subfield of the FIB$W_ALLOC field to specify the volume to which the blocks are moved. If you do not specify a volume, the blocks are moved to the volume containing the first virtual block. Note that you cannot move blocks of a directory file to another volume.

If the file was previously marked contiguous, the movefile operation sets the FIBV_ALCON, FIBV_ALCONB, and FIB$V_FILCON flags. This ensures that a contiguous file is not fragmented by a movefile operation.

For virtual blocks beyond the file's highwater mark, the movefile operation allocates new logical blocks but does not copy the contents. The position of the file's highwater mark remains unchanged.

1.6.7 Mount

On VAX and Alpha systems, 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$_MOUNT takes no arguments or function modifiers. This function is part of the volume mounting operation only, and it is not meant for general use. Most of the actual processing is performed by the MOUNT command or the Mount Volume ($MOUNT) system service.

1.6.8 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$M_DMOUNT—Dismounts a volume.

1.6.8.1 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).

Table 1–14 lists FIB fields that control the processing of the IO$_ACPCONTROL function.

Table 1–14 IO$_ACPCONTROL and the File Information Block

Field	Subfields	Meaning
FIB$W_CNTRLFUNC		Specifies the control function to be performed. This field overlays FIB$W_EXCTL.
FIB$L_CNTRLVAL[1]		Specifies additional function-dependent data. This field overlays FIB$L_EXSZ.
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. For Files–11 C/D, this field is always set to SS$_NORMAL.
FIB$L_STATUS[1]		Alternate access status. The following bits are supported.
	FIB$V_ALT_REQ	Set to indicate whether the alternate access bit is required for the current operation. If not set, the alternate access bit is optional.
	FIB$V_ALT_GRANTED	If FIB$V_ALT_REQ = 0 and the alternate access check succeeded, the FIB bit returned from the file system is set.
FIB$L_ALT_ACCESS[1]		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.

[1]Not supported or invalid for Files–11 C/D.

1.6.8.2 Magnetic Tape Control Functions

Table 1–15 the lists FIB field applicable to magnetic tape operations.

Table 1–15 Magnetic Tape Operations and the File Information Block

Field	Subfields	Meaning
FIB$W_CNTRLFUNC		Several ACP control functions are used for magnetic tape positioning. These functions are specified by supplying a FIB with P1 containing the FIB descriptor address. Modifiers and parameters P2, P3, and P4 are not allowed. These functions clear serious exceptions in magnetic tape drivers. The following control functions can be specified to control magnetic tape positioning.
	FIB$C_REWINDFIL	Rewind to beginning-of-file.
	FIB$C_REWINDVOL	Rewind to beginning-of-volume set.
	FIB$C_POSEND	Position to end-of-volume set.
	FIB$C_NEXTVOL	Force next volume.
	FIB$C_SPACE	Space *n* blocks forward or backward. The FIB$L_CNTRLVAL field specifies the number of magnetic tape blocks to space forward if positive or to space backward if negative.
	FIB$C_CLSEREXCP	If set, clears the serious exception in the magnetic tape driver (see FIB$C_USEREOT in Section 1.6.1 and Section 1.6.2). If writing, 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). If reading, this allows the user to handle the move to the next volume or to stop reading the tape. The user should not attempt to read past EOV.

1.6.8.3 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$M_DMOUNT — Specifying the dismount modifier on the IO$_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.

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$C_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$C_LOCK_VOL — Allocation lock the volume. Operations that change the file structure, such as file creation, deletion, extension, and deaccess, are not permitted. If such requests are queued to the file system for an allocation-locked volume, they are not processed until the FIB$C_UNLK_VOL function is issued to unlock the volume.

To issue the FIB$C_LOCK_VOL function, you must have either a system UIC or SYSPRV privilege, or be the owner of the volume.

FIB$C_UNLK_VOL	Unlock the volume. Cancels FIB$C_LOCK_VOL. To issue this function, you must have either a system UIC or SYSPRV privilege, or be the owner of the volume.

1.6.8.4 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 1–16 lists the enable and disable quota control functions.

Table 1–16 Disk Quota Functions (Enable/Disable)

Value	Meaning
FIB$C_ENA_QUOTA	Enable the disk quota file. If a nonzero directory file ID is specified in FIB$W_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$W_FID, if present, is accessed by the ACP, and quota enforcement is turned on. By convention, the quota file is named [0,0]QUOTA.SYS;1. Therefore, FIB$W_DID should contain the value 4,4,0 and the name string specified with P2 should be ''QUOTA.SYS;1''.
FIB$C_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 1–17 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.

Table 1–17 Disk Quota Functions (Individual Entries)

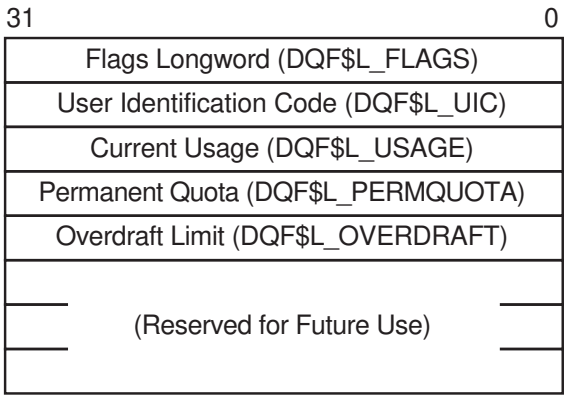
Value	Meaning
FIB$C_ADD_QUOTA	Add an entry to the disk quota file, using the UIC and quota specified in the P2 argument block. FIB$C_ADD_QUOTA requires write access to the quota file.
FIB$C_EXA_QUOTA	Examine a disk quota file entry. The entry whose UIC is specified in the P2 argument block is returned in the P4 argument block, and its length is returned in the P3 argument word. Using two flags in FIB$L_CNTRLVAL, it is possible to search through the quota file using wildcards. The two flags are: FIB$V_ALL_MEM — Match all UIC members FIB$V_ALL_GRP — Match all UIC groups The ACP maintains position context in FIB$L_WCC. On the first examine call, you specify 0 in FIB$L_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.

(continued on next page)

Table 1–17 (Cont.) Disk Quota Functions (Individual Entries)

Value	Meaning
FIB$C_MOD_QUOTA	Modify a disk quota file entry. The quota file entry specified by the UIC in the P2 argument block is modified according to the values in the block, as controlled by three flags in FIB$L_CNTRLVAL.
	FIB$V_MOD_PERM Change the permanent quota
	FIB$V_MOD_OVER Change the overdraft quota
	FIB$V_MOD_USE Change the usage data
	The usage data can be changed only if the volume is locked by FIB$C_ LOCK_VOL (see Section 1.6.8.3). FIB$C_MOD_QUOTA requires write access to the quota file.
	The P3 and P4 arguments return the modified quota entry to you.
	By using the flags FIB$V_ALL_MEM and FIB$V_ALL_GRP, you can search through the quota file using wildcards just as you would with the FIB$C_EXA_QUOTA function.
FIB$C_REM_QUOTA	Remove a disk quota file entry whose UIC is specified in the P2 argument block. FIB$C_REM_QUOTA requires write access to the quota file.
	The P3 and P4 arguments return the removed quota file entry to you.
	By using the flags FIB$V_ALL_MEM and FIB$V_ALL_GRP, you can search through the quota file using wildcards just as you would with the FIB$C_EXAQUOTA function.

Figure 1–9 Quota File Transfer Block



ZK–0643–GE

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.

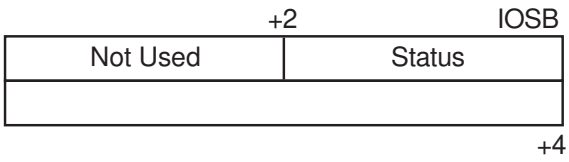
1.7 I/O Status Block

Figure 1–10 shows the I/O status block (IOSB) for ACP–QIO functions. Appendix A lists the status returns for these functions. (The OpenVMS system messages documentation 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$V_ALCON) fails, the second longword is used to return the size of the file after truncation.

Values returned in the IOSB are most useful during operations in compatibility mode. When executing programs in the native mode, use the values returned in FIB locations.

Figure 1–10 IOSB Contents—ACP-QIO Functions



ZK–0644–GE

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
Disk Drivers

This chapter describes the use of disk drivers that support the disk devices listed in the *Software Product Description*. The chapter also includes descriptions of many of the supported disks and controllers. However, not all supported devices are described here. Refer to the OpenVMS *Software Product Description* for the definitive list of supported devices.

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 RMS calls described in the *OpenVMS Record Management Services Reference Manual*. Files in RT–11 format can be read or written with the file exchange facility EXCHANGE.

2.1 Supported Disk Devices and Controllers

The sections that follow provide descriptions of disk devices.

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 2.3 lists the information on disk devices returned by $GETDVI.

2.1.1 UDA50 UNIBUS Disk Adapter

The UDA50 UNIBUS Disk Adapter is a microprocessor-based disk controller for mass storage devices that implements the DIGITAL Storage Architecture (DSA); for more information on the DSA, see Section 2.2.3.

The UDA50 controller 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 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 UDA50 controllers per UNIBUS are supported. If two UDA50 controllers 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 operating system processes request virtual or logical I/O on disks controlled by the UDA50. The 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 controller 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 2.2.11.1). Any bad blocks that might exist on a disk attached to a UDA50 are transparent to the

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

2.1.2 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 2.1.1).

2.1.3 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 2.1.1).

2.1.4 HSC40, HSC50, and HSC70 Controllers

HSC controllers are high-speed, high-availability controllers for mass storage devices that implement the DIGITAL Storage Architecture (DSA); for more information about the DSA, see Section 2.2.3. An HSC controller is connected to a processor by a CI bus (computer interconnect). The operating system supports the use of the HSC controllers in controlling the RA family of disks.

The HSC40 can support up to 12 SDI (standard disk interface) disks from the SA or RA families of disk drives or a combination of up to 12 SDI disk drives and TA-series tape drives.

The HSC70 can support up to 32 SDI disks from the SA or RA families of disk drives or a combination of SDI disk drives and TA-series tape drives.

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

HSC controllers correct 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 operating system, which performs logical or virtual I/O to such a disk, does not recognize that any bad blocks might exist on a disk attached to an HSC controller. HSC controllers also correct most data errors.

The HSC series of controllers provides access to disks despite most hardware failures. Use of an HSC controller permits 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 operating system enables read/write access to all nodes that are members of the same cluster.

HSC-series controllers allow you to add or subtract disks from the device configuration without rebooting the system.

2.1.5 SII Integral Adapter

The SII integral adapter on the MicroVAX 3300/3400 processor provides access through the DIGITAL Storage Systems Interconnect (DSSI) bus to a maximum of seven storage devices.

The term **dual-host** refers to pairs of CPUs connected to a bus. In dual-host configurations of MicroVAX 3300/3400 CPUs, the DSSI bus must be connected between the SII integral adapters present on both CPUs.

A maximum of six devices can be connected to the EDA640 adapter, which is implemeneted by the SII chip, DXX chip, and 128K RAM chip, in dual-host configurations.

2.1.6 KFQSA Adapter

The KFQSA adapter allows a maximum of seven storage devices for use on Q–bus systems.

In dual-host configurations of MicroVAX 3800/3900 CPUs, the DSSI bus must be connected between KFQSA adapters present on both CPUs.

A maximum of six devices can be connected to the KFQSA adapter in dual-host configurations.

2.1.7 RQDX3 Controller

The RQDX3 controller is a Q–bus controller used with the RD series of Winchester-type disk drives and the RX33 and RX50 flexible diskette drives.

2.1.8 RA70 and RA90 Disk Drives

The RA70 is a 5.25-inch 280-megabyte (MB) high-performance DSA disk drive that uses thin-film media. It has an average access time of 27.0 milliseconds (ms) and average seek time of 19.5 ms. The RA70 uses the Standard Disk Interface (SDI) and the KDA50 controller, and can be dual-ported.

The RA90 is a 1.2-gigabyte disk drive designed with thin-film heads and 9-inch thin-film media with an average seek time of 18.5 ms. The RA90 conforms to DSA and uses the SDI. Both the RA70 and RA90 disk drives can be connected to medium-sized systems with the HSC-series controllers, KDB50, or UDA50 controllers.

2.1.9 RA60 Disk

The RA60 device uses high-capacity, removable media that provides 205 MB of usable storage (7.5 million bits of data per square inch) with transfer rates of 1.9 MB per second (burst) and 950 kb per second (sustained). The RA60 belongs to the DIGITAL Storage Architecture (DSA) family of disk devices (see Section 2.2.3). It is connected to either 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.

2.1.10 RA80/RB80/RM80 and RA81 Fixed-Media Disks

The R80 disk drive is a high-capacity, moving-head disk whose nonremovable media 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.

VAX

- RB80—On VAX systems, 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—On VAX and Alpha systems, 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 high-capacity disk drive with nonremovable media 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.

The RA80 and RA81 belong to the DIGITAL Storage Architecture (DSA) family of disk devices (see Section 2.2.3).

2.1.11 RB02 and RL02 Cartridge Disk (VAX Only)

VAX

On VAX systems, 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 2.4).

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.♦

2.1.12 RC25 Disk (VAX Only)

VAX

On VAX systems, 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 a Q–bus system.♦

2.1.13 RD53 and RD54 Disks (VAX Only)

VAX

On VAX systems, the RD53 and RD54 are 5.25-inch, full-height, Winchester-type drives with average access time of 38 ms and a data transfer rate of 0.625 MB per second. The RD53 and RD54 have a formatted capacity of 71 MB and 159 MB, respectively. When used with the RQDX3 controller, the RD53 and RD54 are DSA disks.

See Section 2.2.12 for information about using RD series disks on the VAXstation 2000.♦

2.1.14 RF30 and RF71 Disks

The RF30 is a 150-MB, 5.25-inch, half-height disk drive while the RF71 is a 400-MB full-height disk drive. The RF30 and RF71 include an embedded controller for multihost access and a mass storage control protocol (MSCP) server. The RF71 has a peak data transfer rate of 1.5 MB per second with average seek and access time of 21 ms and 29 ms, respectively.

Both the RF30 and RF71 disks use DIGITAL Storage System Interconnect (DSSI) bus and host adapters.

2.1.15 RK06 and RK07 Cartridge Disks (VAX Only)

VAX

On VAX systems, 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.♦

2.1.16 RM03 and RM05 Pack Disks (VAX Only)

VAX

On VAX systems, 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.♦

2.1.17 RP05 and RP06 Disk (VAX Only)

VAX

On VAX systems, the RP05 and RP06 removable disks consist of 19 data surfaces and a moving read/write head. The RP06 removable 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.♦

2.1.18 RP07 Fixed Media Disk (VAX Only)

VAX

On VAX systems, 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 rev/min and their data is accessed at an average speed of 31.3 ms. These disks are connected to the system by an MBA. Up to eight disk drives can be connected to each MBA.♦

2.1.19 RRD40 and RRD50 Read-Only Memory (CD–ROM)

The RRD40 and RRD50 are compact disc read-only memory (CD–ROM) devices that use replicated media with a formatted capacity of approximately 600 MB.

The RRD40 is a 5.25-inch half-height, front-loading tabletop or embedded device that attaches to the system using either the Small Computer System Interface (SCSI) or Q–bus interface.

The RRD50 is a 5.25-inch, top-loading tabletop device that attaches to the system using a Q–bus interface.

The RRD40 has an average access time of 0.5 second while the average access time for the RRD50 is 1.5 seconds. Both the RRD40 and RRD50 have a data transfer rate of 150 KB per second.

The media for the RRD40 and the RRD50 are removable 4.7-inch (120-mm) compact discs. However, the media for the RRD40 are enclosed in protective self-loading carriers. The RRD40 with a SCSI interface is also available as an embedded unit. The RRD40 and RRD50 Q–bus subsystems are standard disk MSCP devices.

2.1.20 RX01 Console Disk (VAX Only)

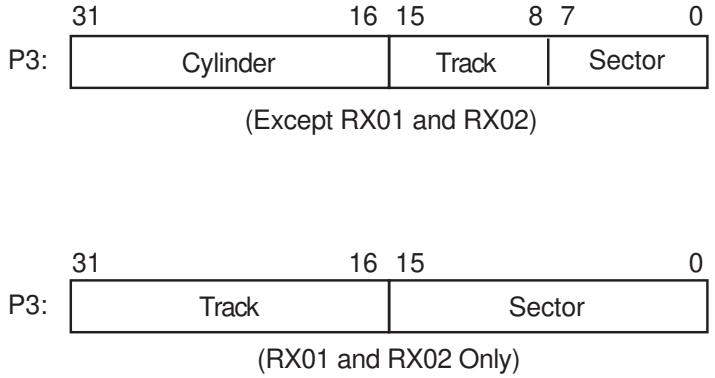
VAX

On VAX systems, 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 2.2.10 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 2–1 and Section 2.4). Note that the driver does not apply track-to-track skew, cylinder offset, or sector interleaving to this physical medium address.♦

Figure 2–1 Disk Physical Address



2.1.21 RX02 Disk (VAX Only)

VAX

On VAX systems, 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 2.2.10 describes this feature in greater detail.

For physical block I/O operations, the track and sector parameters shown in Figure 2–1 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.♦

2.1.22 RX23 (VAX Only)

VAX

On VAX systems, the RX23 device is a 1 inch high, flexible diskette drive that uses 3.5-inch microfloppy diskettes. The RX23 drive can access standard- and high-density media. The following table summarizes capacities for standard- and high-density media.

Density	Unformatted	Formatted
Standard	1.0 MB	700 KB
High	2.0 MB	1.4 MB

The RX23 is backwardly compatible in that it can read 1-MB media. It can also read and write 2.0-MB double-sided, high-density (135 tracks per inch) media.

The RX23 communicates with the controller using the ST506 fixed-disk interconnect (FDI).♦

2.1.23 RX33 (VAX Only)

VAX

On VAX systems, the RX33 is a 1.2-MB, 5.25-inch, half-height diskette drive. The RX33 can record in either standard- or high-density mode. High-density mode provides 1.2 MB of storage using 96 tracks per inch using double-sided, high-density diskettes.

In standard-density mode, the RX33 drive is read- and write-compatible with single-sided, standard-density RX50 diskettes.♦

2.1.24 RX50 (VAX Only)

VAX

On VAX systems, the RX50 dual-diskette drive stores data in fixed-length blocks on 5.25-inch 0.8-MB, flexible diskettes using preformatted headers. The RX50 can accommodate two diskettes simultaneously.♦

2.1.25 RZ22, RZ23, and RZ55 Disks

The RZ22 and RZ23 are 3.5-inch, half-height SCSI drives with an average seek rate of 33 ms and an average data transfer rate of 1.25 MB per second. The RZ22 and RZ23 have capacities of 52 MB and 104 MB, respectively.

The RZ55 is a 332-MB, 5.25-inch, full-height SCSI drive with an average access rate of 24 ms.

2.1.26 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 TU58, which has its own controller, can access either one or two tape drives.

2.2 Driver Features

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

The following sections describe these features in greater detail.

2.2.1 Dual-Pathed Disks

A **dual-pathed disk** is a dual-ported disk that is accessible to all the CPUs in the cluster, not just to the CPUs that are connected physically to the disk. Dual-pathed disks can be any of the following:

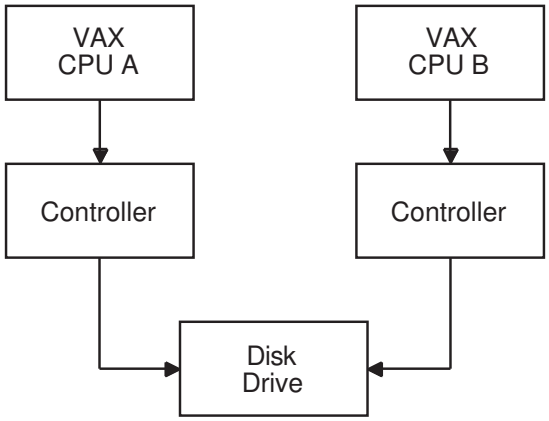
- Dual-ported MASSBUS disks
- Dual-ported HSC disks
- Dual-pathed DSA disks on local UDA50, KDA50, and KDB50 controllers
- Dual-ported RF-series disks

The term dual-pathed refers to the two paths through which clustered CPUs can access a disk to which they are not directly connected. If one path fails, the disk is accessed over the other path. (Note that with a dual-ported MASSBUS disk, a CPU directly connected to the disk always accesses it locally.)

2.2.2 Dual Porting MASSBUS Disks

The MASSBUS disk drivers, DBDRIVER and DRDRIVER, support static dual porting. Dual porting allows two MASSBUS controllers to access the same disk drive. Figure 2–2 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 2–2 Dual-Ported Disk Drives



ZK–0650–GE

2.2.2.1 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.
  - 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 1-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.

2.2.2.2 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.

2.2.2.3 Restriction on Dual-Ported Non-DSA Disks in a Cluster

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 cluster, 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.
- 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 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 System Generation 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.

2.2.3 Dual-Pathed DSA Disks

A dual-ported DSA disk can be failed over between the two CPUs that serve it to the cluster under the following conditions: (1) the same disk controller letter and allocation class are specified on both CPUs and (2) both CPUs are running the MSCP server.

Caution

Failure to observe these requirements can endanger data integrity.

However, because a DSA disk can be online to only one controller at a time, only one of the CPUs can use its local connection to the disk. The second CPU accesses the disk through the MSCP server. If the CPU that is currently serving the disk fails, the other CPU detects the failure and fails the disk over to its local connection. The disk is thereby made available to the cluster once more.

Note

A dual-ported DSA disk may not be used as a system disk.

2.2.4 Dual-Porting HSC Disks

By design, HSC disks are cluster accessible. Therefore, if they are dual ported, they are automatically dual pathed. CI-connected CPUs can access a dual-pathed HSC disk by way of a path through either HSC-connected device.

For each dual-ported HSC disk, you can control failover to a specific port using the port select buttons on the front of each drive. By pressing either port select button (A or B) on a particular drive, you can cause the device failover to the specified port.

With the port select button, you can select alternate ports to balance the disk controller workload between two HSC subsystems. For example, you could set half of your disks to use port A and set the other half to use port B.

The port select buttons also allow you to failover all the disks to an alternate port manually when you anticipate the shutdown of one of the HSC subsystems.

2.2.5 Dual-Pathed RF-Series Disks

In a dual-path configuration of MicroVAX 3300/3400 CPUs or MicroVAX 3800 /3900 CPUs using RF-series disks, CPUs have concurrent access to any disk on the DSSI bus. A single disk is accessed through two paths and can be served to all satellites by either CPU.

If either CPU fails, satellites can access their disks through the remaining CPU. Note that failover occurs in the following situations: (1) when the DSSI bus is connected between SII integral adapters on both MicroVAX 3300/3400 CPUs or (2) when the DSSI bus is connected between the KFQSA adapters on pairs of MicroVAX 3300/3400s or pairs of MicroVAX 3800/3900s.

Note

The DSSI bus should not be connected between a KFQSA adapter on one CPU and an SII integral adapter on another.

2.2.6 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 *OpenVMS DCL Dictionary* describes volume mounting and dismounting. The *OpenVMS 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 *OpenVMS Record Management Services Reference Manual* describe file access.

Offset recovery is performed during a data check but error code correction (ECC) correction is not performed (see Section 2.2.9). 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 completed successfully, 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 2.4.1 and 2.4.2).

2.2.7 Effects of a Failure During an I/O Write Operation

The operating system ensures that when an I/O write operation returns a successful completion status, the data is available on the disk or tape media. Applications that must guarantee the successful completion of a write operation can verify that the data is on the media by specifying the data check function modifier IO$M_DATACHECK. Note that the IO$M_DATACHECK data check function, which compares the data in memory with the data on disk, affects performance because the function incurs the overhead of an additional read operation to the media.

If a system failure occurs while a multiple-block write operation is in progress, the operating system does not guarantee the successful completion of the write operation. (OpenVMS does guarantee single-block write operations to DSA drives.) When a failure interrupts a write operation, the data may be left in any one of the following conditions:

- The new data is written completely to the disk blocks on the media, but a completion status was not returned before the failure.
- The new data is partially written to the media so that some of the disk blocks involved in the I/O contain the data from the write operation in progress, and the remainder of the blocks contain the data that was present before the write operation.
- The new data was never written to the disk blocks on the media.

To guarantee that a write operation either finishes successfully or (in the event of failure) is redone or rolled back as if it were never started, use additional techniques to assure data correctness and recovery. For example, using database journaling and recovery techniques allows applications to recover automatically from failures such as the following:

- Permanent loss of the path between a CPU data buffer containing the data being written and the disk being written to during a multiple-block I/O operation. Communication path loss can occur due to node or controller failure or a failure of node-to-node communications.
- Failure of a CPU (such as a system failure, system halt, power failure, or system shutdown) during a multiple-block write operation.
- Mistaken deletion of a file.
- Corruption of file system pointers.
- File corruption due to a software error or incomplete bucket write operation to an indexed file.
- Cancellation of an in-progress multiple-block write operation.

2.2.8 Overlapped Seeks

A seek operation involves moving the disk read/write heads to a specific place on the disk without any transfer of data. All transfer functions, including data checks, are preceded by an implicit seek operation (except when the seek is inhibited by the physical I/O function modifier IO$M_INHSEEK). Seek operations can be overlapped *except* on RL02, RX01, RX02, TU58 drives, MicroVAX 2000, VAXstation 2000, or on controllers with floppy disks (for example, RQDX3) when the disk is executing I/O requests. 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.

2.2.9 Error Recovery

Error recovery in the 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 and SCSI disks, this function is implemented by the controller.
- Applying error correction information (not applicable for RB02, RL02, RX01, RX02, and TU58). For DSA and SCSI 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 and SCSI disks, this function is implemented by the controller.

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.

2.2.9.1 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.

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

Logical-block-to-physical-sector translation on RX01 and RX02 drives adheres to the standard 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.

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 DEC Fortran for OpenVMS 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
```

2.2.11 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 a drive and its controller
- Controller-to-host communications—Describes how hosts request controllers to transfer data

Because the 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, KDA50, KDB50, 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.

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 revectored 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 2.2.11.1 for additional information about bad block replacement for DSA disks.
- Insert a WAIT statement in your SYSTARTUP_V5.COM file on VAX systems, or your SYSTARTUP_VMS.COM file on Alpha systems, prior to the first MOUNT statement for a DSA disk. The wait period should be about 2 to 4 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 2.2.3).
- In many cases, you can attach a DSA disk to its controller on a running system and then use it without manual intervention.
- DSA disks and the DUDRIVER do not accept physical QIO data transfers or seek operations.

2.2.11.1 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 (revectored) to one of the replacement blocks. The process that revectors logical blocks is called a **bad block 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.

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 revectored to a replacement block. After the block is revectored, the original data is copied back to the revectored 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 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.

2.2.12 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.
- Error messages occur 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 Customer Services.

2.2.13 SCSI Disk Class Driver

The VAXstation 3100, 3520, and 3540 contain a SCSI bus that provides access to as many as seven SCSI disks. The SCSI disk class driver controls SCSI disks on all of the above systems. Although, SCSI disks do not conform to DSA, they do support the following error recovery features:

- Static and dynamic bad block replacement (BBR)
- Error correction code (ECC)
- Reexecution of read or write operations within the SCSI drive
- Reexecution of read or write operations by the SCSI disk class driver

All SCSI disks supplied by Digital implement the REASSIGN BLOCKS command which relocates data for a specific logical block to a different physical location on the disk. The SCSI disk class driver reassigns the block in the following instances: (1) when the retry threshold is exceeded during an attempt to read or write a block of data on the disk or (2) when an irrecoverable error occurs during a write operation.

Unlike DSA, there is no forced error flag in SCSI. Blocks that produce irrecoverable errors during read operations are not reassigned in order to prevent undetected loss of user data. Instead, the SCSI disk class driver returns the SS$_PARITY status whenever a read operation results in an irrecoverable error.

2.2.14 Audio Extensions to the SCSI Disk Class Driver

This section describes SCSI disk class driver audio commands and the $QIO interface by which the operating system provides audio functionality to the SCSI disk.

Table 2–1 lists the SCSI audio commands supported by the SCSI disk class driver.

Table 2–1 SCSI Disk Class Driver Audio Commands

Command	Audio Function Code[1]	Description
Play Audio MSF	AUDIO_PLAY_AUDIO_MSF (5)	Requests the CD–ROM to begin an audio playback operation. The two required command arguments specify absolute starting and ending addresses of the playback in terms of minutes, seconds, and frame (MSF).
Play Audio Track	AUDIO_PLAY_AUDIO_TRACK (6)	Requests the CD–ROM to begin an audio playback operation. The two required command arguments specify the starting and ending tracks of the playback in terms of track number and index.
Play Audio	AUDIO_PLAY_AUDIO (4)	Requests the CD–ROM to begin an audio playback operation. The two required command arguments specify the starting logical block address (LBA) and the transfer count, in blocks, of the playback.
Pause	AUDIO_PAUSE (0)	Requests the CD–ROM to suspend any active audio operations. In response, the CD–ROM enters the hold-track state, muting the audio output after playing the current block.
Resume	AUDIO_RESUME (1)	Requests the CD–ROM to resume any active audio operations. In response, the CD–ROM exits the hold-track state and resumes playback at the block following the last block played.
Get Status	AUDIO_GET_STATUS (9)	Requests from the CD–ROM the status of the currently active playback operation, as well as the state of the current block. The Get Status command corresponds to the SCSI II Read Sub-channel command (READ SUBQ)
Set Volume	AUDIO_SET_VOLUME (11)	Requests the CD–ROM to adjust the output channel selection and volume settings for ports 0 through 3. The Set Volume command corresponds to the SCSI II Mode Select command for the CD–ROM Audio Control Parameters page.
Get Volume	AUDIO_GET_VOLUME (12)	Requests from the CD–ROM the output channel selection and volume settings for ports 0 through 3. The Get Volume command corresponds to the SCSI II Mode Sense command for the CD–ROM Audio Control Parameters page.
Prevent Removal	AUDIO_PREVENT_REMOVAL (2)	Prevents the removal of the CD caddy from the CD–ROM drive.

[1]Symbolic values for the function codes of SCSI audio commands are defined in SYS$EXAMPLES:CDVERIFY.C. Numeric values appear within parentheses in this table column.

(continued on next page)

Table 2–1 (Cont.) SCSI Disk Class Driver Audio Commands

Command	Audio Function Code[1]	Description
Allow Removal	AUDIO_ALLOW_REMOVAL (3)	Allows the removal of the CD caddy from the CD–ROM drive.
Get TOC	AUDIO_GET_TOC (10)	Requests from the CD–ROM a list of each track on the disk, including information about the audio or data contents of each track. Applications that require a detailed knowledge of the organization of a CD–ROM can use this function to obtain that information. The Get TOC command corresponds to the SCSI II Read TOC command.

[1]Symbolic values for the function codes of SCSI audio commands are defined in SYS$EXAMPLES:CDVERIFY.C. Numeric values appear within parentheses in this table column.

2.2.14.1 $QIO Interface to Audio Functionality of the SCSI Disk Class Driver

To employ the audio functions of the RRD42 CD–ROM reader, the application program issues a call to the $QIO system service using the following format:

status=SYS$QIO ([efn] ,[chan] ,func [,iosb] [,astadr] [,astprm] [,p1] [,p2] [,p3] [,p4] [,p5] [,p6])

Arguments

[efn]
[chan]
[iosb]
[astadr]
[astprm]

These arguments apply to the $QIO system service completion, not to device interrupt actions. For an explanation of these arguments, see the description of the $QIO system service in the *OpenVMS System Services Reference Manual*.

func

The IO$_AUDIO function code allows the SCSI disk class driver to process SCSI audio commands.

p1

Address of an audio control block (AUCB). The $QIO system service passes a SCSI audio command and command-specific control information to the SCSI disk class driver in the AUCB structure (see Section 2.2.14.2).

p2

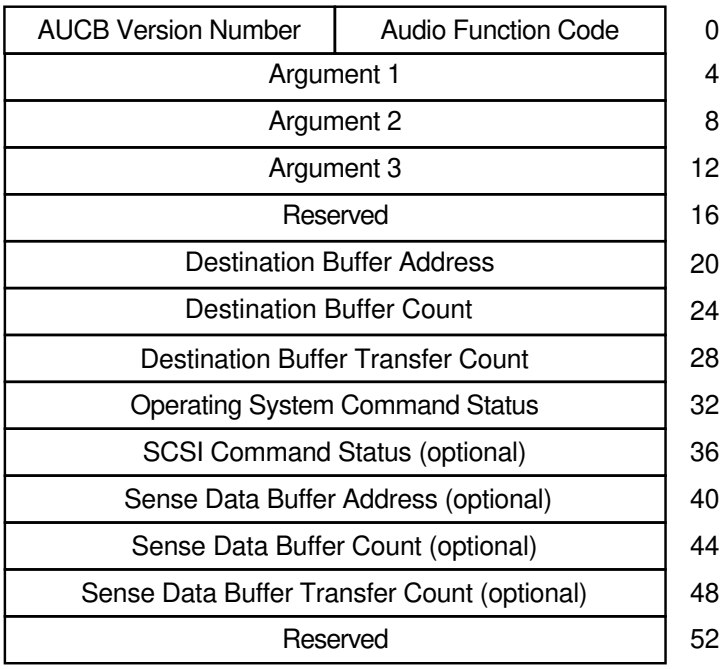
Size of the AUCB.

2.2.14.2 Defining an Audio Control Block (AUCB)

An application program that issues a call to the $QIO system service that specifies the IO$_AUDIO function code in the **func** argument must supply the address of an AUCB structure in the **p1** argument and its size in the **p2** argument.

An AUCB defines a specific SCSI audio command and provides the SCSI disk class driver with command-specific arguments and control information. Table 2–2 defines the fields that appear in an AUCB; these fields are pictured in Figure 2–3. See SYS$EXAMPLES:CDROM_AUDIO.C for a code example that illustrates how an AUCB is defined in the C programming language.

Figure 2–3 Audio Control Block (AUCB)



ZK–4625A

Table 2–2 Contents of Audio Control Block

Field	Use
Audio Function Code	Numeric or symbolic code representing the audio function desired by the application program. (See Table 2–1.) The application program must provide a valid audio function code for each operation.
AUCB Version Number	Version of the AUCB and SCSI disk class driver audio interface. For the current version of the interface the value of this field should be 1. This field must *never* contain a zero.
Argument 1	This field is audio command specific and contains the first argument of the function as follows:

Audio Function Code[1]	Field Contents
AUDIO_PLAY_ AUDIO_MSF (5)	Starting Frames | (Sec shifted left 8 bits) | (Min shifted left 16 bits)
AUDIO_PLAY_ AUDIO_TRACK (6)	Starting (Track shifted left 8 bits) | Index

[1]For any function code not listed in this table, this field contains a zero.

(continued on next page)

Table 2–2 (Cont.) Contents of Audio Control Block

Field	Use	
	AUDIO_PLAY_AUDIO (4)	Starting logical block address
	AUDIO_GET_STATUS (9)	0 if LBA format, 1 if MSF format. See the SCSI II specification for information about these formats.
	AUDIO_SET_ VOLUME (11)	Longword representing the values to be used to determine the new output channel selection and volume settings for CD–ROM ports 0 through 3. See Figure 2–4 for an illustration of the format of this longword. Note that many CD–ROM drives do not support ports 2 and 3.
	AUDIO_GET_ VOLUME (12)	Longword to receive the current values determining output channel selection and volume settings for CD–ROM ports 0 through 3. See Figure 2–4 for an illustration of the format of this longword. Note that many CD–ROM drives do not support ports 2 and 3.
	AUDIO_GET_TOC (10)	0 if LBA format, 1 if MSF format. See the SCSI II specification for information about these formats.
Argument 2	This field is audio command specific and contains the second argument of the function as follows:	
	Audio Function Code[1]	**Field Contents**
	AUDIO_PLAY_ AUDIO_MSF (5)	Starting frames|(sec shifted left 8 bits)|(min shifted left 16 bits)
	AUDIO_PLAY_ AUDIO_TRACK (6)	Ending (track shifted left 8 bits)|index
	AUDIO_PLAY_AUDIO (4)	Transfer count in number of contiguous blocks to be played
	AUDIO_GET_TOC (10)	Starting track
Reserved	Must be zero.	
Destination Buffer Address	Address of the application program's buffer from which the status from a GET_STATUS or GET_TOC function is returned.	
Destination Buffer Count	Size in bytes of the destination buffer specified in the Destination Buffer Address field. For the GET_STATUS function, this field must contain the value 48 to receive complete status information. For the GET_TOC function, this field must contain the value 804 to receive full status. The SCSI disk class driver truncates the status data, if the destination buffer size is smaller than the size of the data.	

[1]For any function code not listed in this table, this field contains a zero.

(continued on next page)

Table 2–2 (Cont.) Contents of Audio Control Block

Field	Use
Destination Buffer Transfer Count	The SCSI disk class driver returns to this field the actual number of bytes transferred to the buffer specified in the Destination Buffer Address field. Before accessing data returned by the GET_TOC or GET_STATUS commands, an application program must check the contents of this field to determine precisely how many bytes were returned by the CD–ROM. The application program initializes this field to zero.
Operating System Command Status	Completion status of the SCSI audio function. This value is also returned in the I/O status block of specified in the **iosb** argument to the $QIO system service call. See Table 2–3 for a description of these status codes. The application program initializes this field to zero.
SCSI Command Status (optional)	SCSI status of the current operation. The SCSI disk class driver returns the SCSI status byte for the SCSI audio command described by this AUCB in the low byte of the low-order word of this field. It returns the sense key in the low byte of the high-order word. See the SCSI II specification for information regarding SCSI status and SCSI sense keys. The application program initializes this field to zero.
Sense Data Buffer Address (optional)	Address of buffer to which the SCSI disk class driver returns sense data when errors occur during audio function execution. When this field is specified, in the event of a check condition on an Audio command, the SCSI disk class driver automatically issues a Request Sense command to retrieve the Sense Key/Sense Data from the target. The target returns this data to the buffer specified in this field before the failing $QIO audio function completes.
Sense Data Buffer Count (optional)	Size in bytes of the buffer specified in the Sense Data Buffer Address field. During request sense processing, the target device truncates the sense data to fit in this buffer.
Sense Data Buffer Transfer Count (optional)	Actual number of bytes of sense data returned to the application in the buffer specified in the Sense Data Buffer Address field. The application program initializes this field to zero.
Reserved	Must be zero.

The output channel selection and volume settings for CD–ROM ports as used by the SET_VOLUME function appear as shown in Figure 2–4.

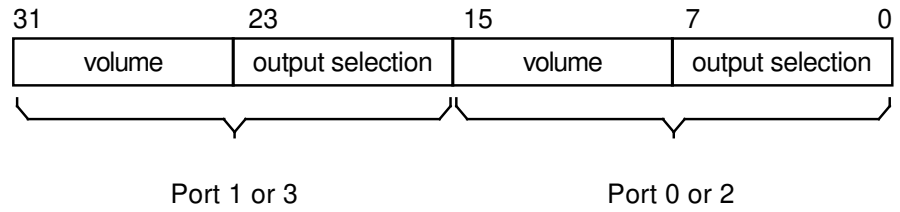
2.2.14.3 Error Handling in Applications Using SCSI Audio Functions

As indicated in Table 2–2, the AUCB provides for three levels of error status reporting:

- Condition values, returned in the Operating System Command Status field of the AUCB, as well as in the I/O status block of specified in the **iosb** argument to the $QIO system service call. (See Table 2–3 for a description of these status codes.)

  If this status is SS$_NORMAL, the function has completed without error. If the status is not SS$_NORMAL, the application program should check the SCSI Command Status field and the Sense Data buffer to fully determine the cause of the failure.

Figure 2–4 Output Channel Selection and Volume Settings for CD–ROM Ports as Used by the SET_VOLUME Function



volume = 00 (muted) to FF (maximum)
output selection <7:4> = 0
output selection <3:0> = 0000 (output muted on this channel)
0001 (connect audio channel 0 to this output port)
0010 (connect audio channel 1 to this output port)
0011 (connect audio channels 0 and 1 to this port)

ZK–4626A

- SCSI command status, returned in the SCSI Command Status field of the AUCB. The SCSI disk class driver returns to this field SCSI status as well as the sense key in the event of a check condition SCSI status. The sense key can be used to determine the first level of error reporting supported by SCSI. See the SCSI II specification for further information.
- Sense data, returned in the buffer specified in the Sense Data Buffer Address field of the AUCB. Sense data bytes are assigned as defined in the SCSI II specification. Sophisticated programmers can use the data in this to obtain detailed information about the error-causing condition.

If the CD–ROM device is currently software-enabled (that is, the volume has been mounted) and a unit attention is detected, then mount verification will be initiated by the driver. However, if the CD–ROM is not software-enabled, the event will simply be returned to the application issuing the Audio $QIO function.

Table 2–3 Status Codes Returned to the IOSB and AUCB by the SCSI Disk Class Driver

SS$_NORMAL	AUCB command completed successfully.
SS$_ABORT	Returned by the SCSI disk port driver. In general, you should retry commands that fail with this status.
SS$_BADPARM	The driver detected an illegal value or missing value in the AUCB.
SS$_CTRLERR	CD–ROM failed some part of its initialization sequence. When this status is returned, it is unlikely that the CD–ROM is usable.
SS$_DEVOFFLINE	Device returned a not-ready sense key or failed the TEST UNIT READY/START sequence.
SS$_DRVERR	CD–ROM failed to execute the command, either because the drive has failed or an illegal command was issued. Such a command could be a command that requested the drive to issue an audio command to a data track or attempted to perform a play operation on nonexistent tracks.
SS$_ILLIOFUNC	Illegal I/O function was specified in the **func** argument of the $QIO request.
SS$_IVADDR	Specified block number is larger than UCB$L_ MAXBLOCK.
SS$_MEDOFL	Last command failed because the drive detected the removal and replacement of the CD carrier, or the unsuccessful completion of a Request Sense command after a check condition error. In general, you should not retry commands that fail with this status.
SS$_NOPRIV	Caller does not have sufficient privileges to execute this function. If the CD–ROM has not been mounted before an IO$_READVBLK function is issued, this status may be returned.
SS$_OPINCOMPL	Number of bytes requested is less than the number of bytes returned.
SS$_PARITY	Nonrecoverable media error (does not apply to audio functions).
SS$_RECOVERR	Recovered media error (does not apply to audio functions).
SS$_VOLINV	CD–ROM has not been mounted.
SS$_WRITLCK	Write operations not permitted on read-only devices.

2.2.14.4 Using CD–ROM to Store Both Data and Audio Information

In order to make effective use of mixed data and audio CDs, an application program requires detailed knowledge of the particular CD being played. The application program must know which tracks include data and which tracks include audio so it can use commands appropriate to the different track types. Issuing an audio command on a data track results in the command failing with a status of SS$_DRVERR.

By default, the SCSI disk class driver transfers all requests issued to a CD–ROM in blocks of 512 bytes. This byte amount is referred to as the default block size. Before attempting to issue READ operations to the CD–ROM, you must ensure that the CD–ROM has been mounted as foreign or as a Files–11 volume. The application program can then determine, by issuing a GET_TOC function, which

tracks (and, therefore, which logical blocks) contain data and which contain audio information.

2.2.14.5 Programming Audio Applications

The following list contains information useful in avoiding problems when writing code using the SCSI audio interfaces:

- If you do not know the type of file system on the CD–ROM, you should mount the CD–ROM as foreign and issue a $QIO request with the logical block I/O read function (IO$_READLBLK) to read individual data blocks. The default block size for all CD–ROMs is 512 bytes.

VAX

- When using the GET_TOC command to obtain CD–ROM address information in LBA format, be advised that the byte ordering of the returned data is in big endian form. Because VAX byte ordering is little endian, you must switch the LBA data bytes to get a logical block address that is valid on a VAX computer. SYS$EXAMPLES:CDROM_AUDIO.C contains an example of how to perform this exchange. ♦
- Before attempting to issue a $QIO request with the virtual block I/O read function (IO$_READVBLK) to the CD–ROM, ensure that the CD–ROM has been mounted. Typically, you have to freign mount non-Files–11 disks. If an IO$_READVBLK $QIO request is issued to an unmounted CD, the request fails with a status of SS$_NOPRIV.

2.2.14.6 Application Program Example Using SCSI Audio Capabilities (VAX only)

VAX

The file SYS$EXAMPLES:CDROM_AUDIO.C contains an example of an application program that performs the following tasks:

- Defines standard symbolic names for the audio function codes representing SCSI audio commands.
- Defines representative AUCBs for each audio function code supported by the SCSI disk class driver.
- Issues a series of $QIO system service requests, each specifying the IO$_ AUDIO function, that exercise the SCSI disk class driver to test its support for CD–ROM drives with audio capabilities.
- Converts LBA data returned by a GET_STATUS command in big-endian byte-ordering form to VAX little-ending byte-ordering form.

♦

2.3 Disk Driver Device Information

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

$GETDVI returns disk characteristics when you specify the item codes DVI$_ DEVCHAR and DVI$_DEVCHAR2. Table 2–4 lists the possible characteristics for disk devices.

Table 2–4 Disk Device Characteristics

Characteristic[1]	Meaning
	Dynamic Bits (Conditionally Set)
DEV$M_AVL	Device is online and available.
DEV$M_CDP[2] ,[3]	Dual-path device with two unit control blocks (UCBs).
DEV$M_CLU[2]	Device is available clusterwide.
DEV$M_2P[2]	Device is dual-pathed.
DEV$M_FOR	Device is foreign.
DEV$M_MNT	Volume is mounted.
DEV$M_RCK	Perform data check all reads.
DEV$M_WCK	Perform data check all writes.
DEV$M_MSCP[2]	Device is accessed using the mass storage control protocol.
DEV$M_RCT	Disk contains replacement and caching table.
DEV$M_SRV[2]	For a cluster, device is served by the MSCP server.
	Static Bits (Always Set)
DEV$M_FOD	Device is file-oriented.
DEV$M_IDV	Device is capable of input.
DEV$M_ODV	Device is capable of output.
DEV$M_RND	Device is capable of random access.
DEV$M_SHR	Device is shareable.

[1]Defined by the $DEVDEF macro.

[2]These bits are located in DVI$_DEVCHAR2.

[3]MASSBUS only.

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 4-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 4-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).

2.4 Disk Function Codes

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.

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 asynchronous system trap (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 2–5 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 2–5 Disk I/O Functions

Function Code	Arguments	Type[1]	Function Modifiers	Function
IO$_ACCESS	P1, [P2],[P3],[P4],[P5]	V	IO$M_CREATE IO$M_ACCESS	Search a directory for a specified file and access the file if found.
IO$_ACPCONTROL	P1,[P2],[P3],[P4],[P5]	V	IO$M_DMOUNT	Perform miscellaneous control functions.
IO$_AVAILABLE		P		Clear volume valid; make DSA units available.
IO$_CREATE	P1,[P2],[P3],[P4],[P5]	V	IO$M_CREATE IO$M_ACCESS IO$M_DELETE	Create a directory entry or a file.
IO$_DEACCESS	P1,[P2],[P3],[P4],[P5]	V		Deaccess a file and, if specified, write final attributes in the file header.
IO$_DELETE	P1,[P2],[P3],[P4],[P5]	V	IO$M_DELETE	Remove a directory entry or file header, or both.
IO$_FORMAT	P1	P		Set density (RX02 only).
IO$_MODIFY	P1,[P2],[P3],[P4],[P5]	V		Modify the file attributes or allocation, or both.
IO$_PACKACK		P		Update UCB fields if RX02; initialize volume valid on other devices. Bring DSA units online.
IO$_READLBLK[6]	P1,P2,P3	L	IO$M_DATACHECK[2] IO$M_INHRETRY	Read logical block.
IO$_READPBLK[6]	P1,P2,P3	P	IO$M_DATACHECK[2] IO$M_INHRETRY IO$M_INHSEEK[3]	Read physical block.[5]
IO$_READVBLK[6]	P1,P2,P3	V	IO$M_DATACHECK[2] IO$M_INHRETRY	Read virtual block.

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

[2] Not for RX01 and RX02.

[3] Not for TU58, RX01, RX02, RB02, and RL02.

[5] Not for DSA and SCSI disks.

[6] On OpenVMS Alpha, P1 supports a 64-bit address.

(continued on next page)

Table 2–5 (Cont.) Disk I/O Functions

Function Code	Arguments	Type[1]	Function Modifiers	Function
IO$_SEARCH	P1	P		Search for specified block or sector (only for TU58).
IO$_SEEK	P1	P		Seek to specified cylinder.[5]
IO$_SENSECHAR		P		Sense the device-dependent characteristics and return them in the I/O status block.
IO$_SENSEMODE		L		Sense the device-dependent characteristics and return them in the I/O status block.
IO$_SETPRFPATH	P1	P	IO$M_FORCEPTH	Specifies a preferred path for DSA disks.
IO$_UNLOAD		P		Clear volume valid; make DSA units available and spin down the volume.
IO$_WRITECHECK[6]	P1,P2,P3	P		Verify data written to disk by a previous write QIO.[2]
IO$_WRITELBLK[6]	P1,P2,P3	L	IO$M_DATACHECK[2] IO$M_ERASE IO$M_INHRETRY	Write logical block.
IO$_WRITEPBLK[6]	P1,P2,P3	P	IO$M_DATACHECK[2] IO$M_ERASE IO$M_INHRETRY IO$M_INHSEEK[3] IO$M_DELDATA[4]	Write physical block.[5]
IO$_WRITEVBLK[6]	P1,P2,P3	V	IO$M_DATACHECK[2] IO$M_ERASE IO$M_INHRETRY	Write virtual block.

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

[2] Not for RX01 and RX02.

[3] Not for TU58, RX01, RX02, RB02, and RL02.

[4] RX02 only.

[5] Not for DSA and SCSI disks.

[6] On OpenVMS Alpha, P1 supports a 64-bit address.

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).
- 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. On OpenVMS Alpha, P1 can be a 64-bit address.
- 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 4-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 4-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. On OpenVMS Alpha, P1 can be a 64-bit address.

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.

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. On OpenVMS Alpha, P1 can be a 64-bit address.
- 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 2–5. (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.

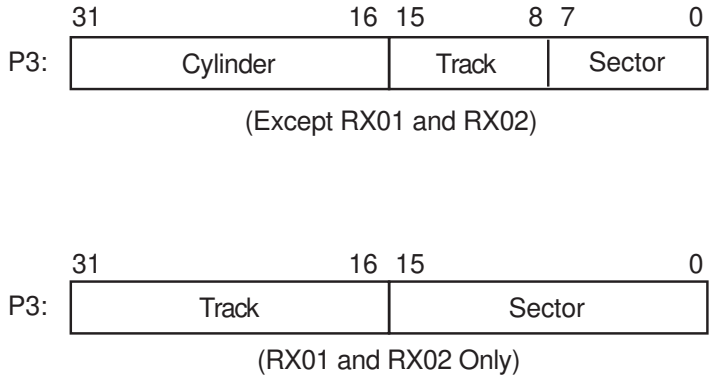
Note

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 2–5.

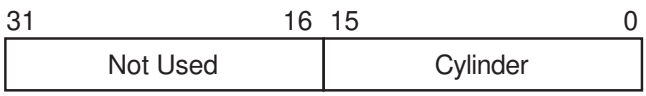
Figure 2–5 Starting Physical Address



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 2–6.

Figure 2–6 Physical Cylinder Number Format



ZK–0653–GE

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

P1—The density at which an RX02 diskette is reformatted (see Section 2.4.4).

2.4.1 Read

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

The 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 2.4.

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.

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

2.4.2 Write

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

The 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 2.4.

The data check function modifier (IO$M_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$M_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$M_INHRETRY) can be used with all write functions. If that modifier is specified, all error recovery attempts are inhibited. IO$M_INHRETRY takes precedence over IO$M_DATACHECK. If both IO$M_INHRETRY and IO$M_DATACHECK are specified and an error occurs, there is no attempt at error recovery, and no data check operation is performed. If an error does not occur, the data check operation is performed. IO$M_INHRETRY has no affect on DSA disks.

The write deleted data function modifier (IO$M_DELDATA) can be used with the write physical block (IO$_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$_WRITEPBLK function is the same; write data is transferred to the disk. When a successful read operation is performed on this data, the status code SS$_RDDELDATA is returned in the I/O status block rather than the usual SS$_NORMAL status code.

The IO$M_ERASE function modifier can be used with all write function codes to erase a user-selected part of a disk. This modifier propagates an erase pattern through the specified range. Section 2.4 describes the write function arguments to be used with IO$M_ERASE.

2.4.3 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 2–8). The 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.

2.4.4 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.

2.4.5 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 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.)

2.4.6 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 locks the drive's port selector on the port that initiated the pack acknowledge function.

In addition, the IO$_PACKACK function updates device-dependent information about DSA disks returned by $GETDVI.

2.4.7 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 2.4.6). The following function code is provided:

IO$_UNLOAD

This function takes no function-dependent arguments.

2.4.8 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 2.4.6). 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.

2.4.9 Seek

The seek function directs the read/write heads to move to the cylinder specified in the P1 argument (see Sections 2.2.8 and 2.4, and Figure 2–6).

2.4.10 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 2–5). 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 2.4.

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.

2.4.11 Set Preferred Path

The set preferred path function specifies a preferred path for DSA disks. This includes RA-series disks and disks accessed through the MSCP server. If a preferred path is specified for a disk, the MSCP disk class drivers (DUDRIVER and DSDRIVER) use the path as their first attempt to locate the disk and bring it on line as a result of a DCL command MOUNT or failover of an already mounted disk. In addition, you can initiate failover of a mounted disk in order to force the disk to the preferred path, or to use load-balancing information for disks accessed through MSCP servers.

The function code is:

IO$_SETPRFPATH

The following is the function modifier:

IO$M_FORCEPATH—Causes the disk class driver to select the server path with the highest load available rating.

The P1 parameter contains the address of a counted ASCII string (.ASCIC). This string is the node name of the HSC or system that is the preferred path. The node name must match an existing node that is known to the local node and if the node is a VAX or Alpha system, it must be running the MSCP server. This function does not move the disk to the preferred path.

The PHYS_IO privilege is required for IO$_SETPRFPATH and IO$M_FORCEPATH.

The following example shows the use of IO$_SETPRFPATH:

```
        $assigndef
        $qiodef
        $iodef
        $exitdef

dev:    .ascid  /$254$DUA48:/

chnl:   .word   0

node:   .ascic  /HSC001/

        .entry  start,0

        $assign_s       devnam=dev,-
                        chan=chnl
        blbc    r0,done
```

```
        $qiow_s         chan=chnl,-
                        func=#IO$_SETPRFPATH,-
                        p1=node

done:
        $exit_s r0

        .end  start
```

This updates the local node I/O database to indicate that node HSC001 is the preferred path for DUA48.

2.4.11.1 Forcing a Path Change

You can move a disk that is already mounted to its preferred path by specifying the IO$M_FORCEPATH modifier. If a preferred path has not been specified for a disk that is accessed through the MSCP server, the IO$M_FORCEPATH function causes the disk class driver to use load-balancing information to select the server path with the highest-load-available rating.

IO$M_FORCEPATH does not accept any arguments. If you intend to move a disk to its preferred path, you must specify the preferred path in a separate $QIO function.

The following example shows use of the IO$M_FORCEPATH function modifier:

```
        $assigndef
        $qiodef
        $iodef
        $exitdef

dev:    .ascid  /$254$DUA197:/

chnl:   .word   0

        .entry  start,0

        $assign_s       devnam=dev,-
                        chan=chnl
        blbc    r0,done

        $qiow_s         chan=chnl,-
                        func=#<IO$_SETPRFPATH!IO$M_FORCEPATH>

done:
        $exit_s r0

        .end    start
```

Note that forcing a path change places the disk in mount verification. New I/O requests are suspended until mount verification is complete.

2.4.11.2 Using IO$_SETPRFPATH with Disks Dual Pathed Between HSCs

You can use the IO$_SETPRFPATH and IO$M_FORCEPATH functions to load balance disks that are dual pathed between HSCs. The IO$M_FORCEPATH function initiates failover of the disk on all nodes that have it mounted and that have a direct path to the HSCs. Since the node that issues the IO$M_ FORCEPATH might not be the first one to attempt failover of the disk, it is essential that all nodes with direct connections to the HSCs specify the same preferred path for the disk. Only one node should issue the IO$M_FORCEPATH request.

2.4.11.3 Using IO$_SETPRFPATH with Disks Dual Pathed Between Systems

You can use IO$M_FORCEPATH to load balance RA-series disks that are dual pathed between systems running the MSCP server. Both serving nodes should specify the same preferred path. In order to move the disk between systems, the system that currently has the disk on line through its local controller should issue the IO$M_FORCEPATH request. The disk must be mounted on both serving nodes.

2.4.11.4 Using IO$_SETPRFPATH with Disks Accessed Through MSCP Servers

You can specify a preferred path for disks that are accessed through MSCP servers. However, this specification overrides any load-balancing decisions.

Note that if a disk can be accessed through both HSC and MSCP servers, you need not specify the HSC as a preferred path. HSC paths are always preferred to server paths.

Using IO$M_FORCEPATH without a preferred path causes the disk class driver to move the disk to the server with the highest available capacity.

2.4.11.5 Using IO$_SETPRFPATH with Phase I Volume Shadowing

You can specify IO$_SETPRFPATH for shadow set members, but not for virtual units. IO$M_FORCEPATH is not supported for shadow set members or virtual units.

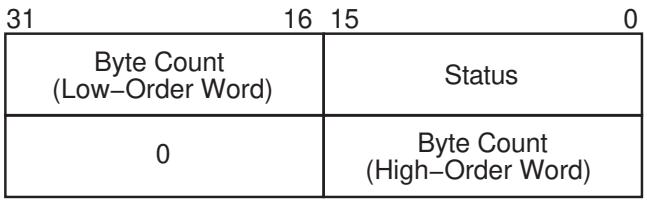
2.4.11.6 Using IO$_SETPRFPATH with Phase II Volume Shadowing

IO$_SETPRFPATH and IO$M_FORCEPATH are supported for shadow set members but not for virtual units.

2.5 I/O Status Block

Figure 2–7 shows the I/O status block (IOSB) for all disk device QIO functions except sense mode. Figure 2–8 shows the I/O status block for the sense mode function. Appendix A lists the status messages for all functions and devices. (The OpenVMS system messages documentation provides explanations and suggested user actions for these messages.)

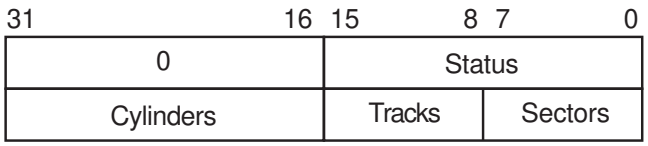
Figure 2–7 IOSB Contents



ZK–0656–GE

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

Figure 2–8 IOSB Contents for the Sense Mode Function



ZK–0657–GE

The second longword of the I/O status block for the sense mode function returns information about the cylinder, track, and sector configurations for the particular device.

2.6 Disk Driver Programming Example

A sample VAX MACRO disk driver program, DISK_DRIVER.MAR, is shown in Example 2–1. This sample program provides an example of optimizing access time to a disk file. The program creates a file using Record Management Services (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.

Example 2–1 DISK_DRIVER.MAR Disk Driver Programming 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
```

(continued on next page)

Example 2–1 (Cont.) DISK_DRIVER.MAR Disk Driver Programming Example

```
;
; 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
;
; 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                            ;
```

(continued on next page)

Example 2–1 (Cont.) DISK_DRIVER.MAR Disk Driver Programming Example

```
;
; 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.
;
IO_STATUS:                              ;
        .BLKQ   1                       ;
;
; Allocate output record buffer.
;
```

(continued on next page)

Example 2–1 (Cont.) DISK_DRIVER.MAR Disk Driver Programming Example

```
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.
;
```

(continued on next page)

Example 2–1 (Cont.) DISK_DRIVER.MAR Disk Driver Programming Example

```
        $PUT    RAB = RAB_BLOCK         ;Put record in file
        BLBC    R0,30$                  ;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
        $CLOSE  FAB = FAB_BLOCK         ;Close file
        BLBS    R0,PART_2               ;If low bit set, successful
                                        ;close
20$     RET                             ;Return with RMS error status
;
; Record stream connection or put record failure.
;
; Close file and return status.
;
30$:    PUSHL   R0                      ;Save error status
        $CLOSE  FAB = FAB_BLOCK         ;Close file
        POPL    R0                      ;Retrieve error status
        RET                             ;Return with RMS error status
;
; The second part of the example illustrates accessing the previously
; created file directly using the Queue I/O system service, randomly
; reading and writing various parts of the file, and then deaccessing
; the file.
;
; First, assign a channel to the appropriate device and access the
; file.
PART_2:                                 ;
        $ASSIGN_S DEVNAM = DEVICE_DESCR,-  ;Assign a channel to file
                CHAN = DEVICE_CHANNEL   ;device
        BLBC    R0,20$                  ;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$M_ACCESS,- ;I/O function is
                -                       ;access file
                IOSB = IO_STATUS,-      ;Address of I/O status
                -                       ;quadword
                P1 = FIB_DESCR          ;Address of information block
                                        ;descriptor
        BLBC    R0,10$                  ;If low bit = 0, access
                                        ;failure
        MOVZWL  IO_STATUS,R0            ;Get final I/O completion
                                        ;status
```

(continued on next page)

Example 2–1 (Cont.) DISK_DRIVER.MAR Disk Driver Programming Example

```
        BLBS    R0,30$                  ;If low bit set, successful
                                        ;I/O function
10$:    PUSHL   R0                      ;Save error status
        $DASSGN_S CHAN = DEVICE_CHANNEL ;Deassign file device channel
        POPL    R0                      ;Retrieve error status
20$:    RET                             ;Return with I/O error status
;
; The file is now ready to be read and written randomly.  Since the
; records are fixed length and exactly one block long, the record
; number corresponds to the virtual block number of the record in the
; file. Thus a particular record can be read or written simply by
; specifying its record number in the file.
;
; The following code reads 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.
;
```

(continued on next page)

Example 2–1 (Cont.) DISK_DRIVER.MAR Disk Driver Programming 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
```

3
Magnetic Tape Drivers

This chapter describes the use of magnetic tape drivers, drives, and controllers.

3.1 Magnetic Tape Controllers and Drives

The sections that follow describe magnetic tape controllers and drives. However, note that not all supported devices are described here. Refer to the OpenVMS *Software Product Description* for the definitive list of supported devices.

3.1.1 TM03 Magnetic Tape Controller (VAX Only)

VAX

On VAX systems, the TM03 magnetic tape controller supports up to eight TE16, TU45, or TU77 tape drives. These dual-density (800 or 1600 bit/inch) 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.♦

3.1.2 TS11 Magnetic Tape Controller (VAX Only)

VAX

On VAX systems, 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-bit/inch, phase-encoded recording.

The TSU05 and the TSV05 magnetic tape drives are used with UNIBUS and Q–bus systems, respectively.♦

3.1.3 TM78 and TM79 Magnetic Tape Controllers (VAX Only)

VAX

On VAX systems, the TM78 and TM79 magnetic tape controllers support up to four TU78 tape drives. These high-performance, dual-density drives (1600 or 6250 bit/inch) 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 bit/inch) mode. The TM78 and TM79 controllers are connected to the MASSBUS through a MASSBUS adapter.♦

3.1.4 TU80 Magnetic Tape Subsystem (VAX Only)

VAX

On VAX systems, the TU80 is a single-density, dual-speed (25 or 100 ips) magnetic tape subsystem that uses streaming tape technology (see Section 3.2.7). It supports one drive per subsystem. The TU80 connects to the UNIBUS through a UNIBUS adapter and completely emulates the TS11 magnetic tape controller.♦

3.1.5 TA81 Magnetic Tape Subsystem

On VAX and Alpha systems, the TA81 is a high-performance, dual-density (1600 or 6250 bit/inch), dual-speed (25 or 75 ips) magnetic tape subsystem that uses streaming tape technology (see Section 3.2.7). It attaches to an HSC50 controller, and is managed with the TMSCP control protocol for tape mass storage.

3.1.6 TU81 Magnetic Tape Subsystem (VAX Only)

VAX

On VAX systems, the TU81 is a high-performance, dual-density (1600 or 6250 bit/inch), dual-speed (25 or 75 in/s) magnetic tape subsystem that uses streaming tape technology (see Section 3.2.7). It connects to the UNIBUS through a UNIBUS adapter, and is managed with the TMSCP control protocol for tape mass storage. ♦

3.1.7 TU81-Plus Magnetic Tape Subsystem (VAX Only)

VAX

On VAX systems, the TU81-PLUS is an enhanced version of the TU81 streaming tape subsystem. It is a 9-track, dual-speed, dual-density, ANSI-standard, half-inch magnetic tape subsystem. In addition, it has a 256-kilobyte (kB) cache buffer that temporarily stores commands and data moving to and from the tape unit. The buffer increases the amount of time the tape drive is able to stream, thereby increasing performance. The TU81-PLUS connects to all VAXBI, UNIBUS, and Q–bus systems using the KLESI–B, KLESI-U, and KLESI-Q adapters.♦

3.1.8 TA90 Magnetic Tape Subsystem

On VAX and Alpha systems, the TA90 is a 5- by 4-inch, 200-MB cartridge tape, fully read- and write-compatible with the IBM 3480 format. The TA90 includes a master controller and a dual transport unit. As many as three additional dual transport slave units can be connected to a single TA90 master controller for a total of eight drives. The controller connects to the HSC 5X-DA high-speed channel card in the HSC.

TA90 tape drives can be equipped with optional stack loaders for unattended backup operations. Each TA90 master has two dual-port STI connections to the HSC. Such dual pathing allows each control unit to service two HSC controllers which significantly increases tape drive availability. The TA90 subsystem includes a 2-MB cache which allows the controller to prefetch upcoming commands and store them while completing current data transfers. This behavior helps optimize performance. The TA90 is a TMSCP device.

3.1.9 RV20 Write-Once Optical Drive (VAX Only)

VAX

On VAX systems, the RV20, a 2-gigabyte, double-sided, write-once optical (WORM) disk drive, is accessed sequentially similar to a tape. A 100-bit error correction code (ECC) protects user data. The controller performs bad block replacement. Three RV20 slaves can be daisy-chained to the subsystem controller in the RV20 master for a total of four drives.

RV02 cartridges can be used on any DIGITAL RV20 optical subsystem.

The average access time is 212.5 ms with an average seek rate of 150 ms. The maximum data transfer rate is 262 kB per second (formatted and sustained) with a burst rate of 1.33 MB per second.♦

3.1.10 TK50 Cartridge Tape System (VAX Only)

VAX

On VAX systems, the TK50 is a 95-MB, 5.25-inch cartridge tape system that uses streaming tape technology (see Section 3.2.7). 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 cyclic redundancy check (CRC) 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. The TZK50 is a SCSI controller for the TK50 tape. Both the TQK50 and the TUK50 are TMSCP devices.

Section 3.1.13 describes compatibility among the TK50, TK70, and TZ30 magnetic tape cartridge systems.♦

3.1.11 TK70 Cartridge Tape System (VAX Only)

VAX

On VAX systems, the TK70 is a 295-MB, 5.25-inch, streaming cartridge tape system. (See Section 3.2.7 for information about streaming tape technology.) The TK70 tape drive records data serially on 48 tracks using serpentine recording, rather than separate (parallel) tracks. Data written to the 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 TQK70 is a dual-height, Q–bus controller for the TK70 magnetic tape drive. The TK70 subsystem includes a 38-kB cache to optimize performance. The TBK70 is a VAXBI-bus controller for the same drive. Section 3.1.13 describes compatibility between the TK50 and TK70 magnetic tape cartridge systems.♦

3.1.12 TZ30 Cartridge Tape System

On VAX and Alpha systems, the TZ30 is a 95-MB, 5.25-inch, half-height cartridge streaming tape drive with an embedded SCSI controller. See Section 3.2.7 for information about streaming tape technology. The TZ30 uses TK50 cartridge tapes. It records data serially on 22 tracks using serpentine recording. Section 3.1.13 describes compatibility between the TK50, TK70, and TZ30 magnetic tape cartridge systems.

3.1.13 Read and Write Compatibility Between Cartridge Tape Systems

When you insert a cartridge tape into the TZ30, TK50, and TK70 tape drives, the hardware initializes the media to a device-specific recording density automatically.

Depending on the type of cartridge and the type of drive on which it is formatted (inserted and initialized), full read and write access to tape cartridges may not be permitted.

Formatting a Blank TK50 Cartridge Tape

A blank, unformatted TK50 cartridge can be formatted on the TK50, TK70, and TZ30 cartridge systems. For example, a TK70 tape drive has full read and write access to a TK50 cartridge formatted on a TK70 drive. Once the cartridge tape is formatted on a particular tape drive, the tape drive has full read and write access to the cartridge tape.

Formatting a Previously Initialized TK50 Cartridge Tape

If a TK50 cartridge tape is formatted on a TZ30 or TK50 cartridge tape drive, the TZ30 and TK50 drives initialize the TK50 cartridge to TK50 density. The following table summarizes the types of access available:

	TK50	
Controller	**Read**	**Write**
TZ30[1]	Yes	Yes

[1]Has an internal controller.

	TK50	
Controller	**Read**	**Write**
TQK50	Yes	Yes
TQK70	Yes	No

The TK70 tape drive can read data on a TK50 cartridge formatted on a TK50 or TZ30 tape drive.

Formatting a TK50 Cartridge Tape on a TK70 Tape Drive

If a TK50 or TK52 cartridge tape is formatted on a TK70 tape drive, the TK70 cartridge tape drive initializes the TK50 or TK52 cartridge tape to TK70 density. The following table summarizes the types of access available:

	TK50		TK52	
Controller	**Read**	**Write**	**Read**	**Write**
TZ30[1]	No	No	No	No
TQK50	No	No	No	No
TQK70	Yes	Yes	Yes	Yes

[1]Has an internal controller.

The TK50 and TZ30 tape drives cannot read or write data on a TK50 cartridge tape formatted on a TK70 drive.

3.2 Driver Features

The 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 TZ30 and TK50 on TUK50 and TQK50 controllers)
- Reverse data check function (except for TZ30, 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

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

3.2.1 Dual Path Tape Drives

A **dual-path** HSC tape drive is a drive that connects to two HSCs, both of which have the same nonzero tape allocation class. The operating system recognizes the dual-pathed capability of such a tape drive under the following circumstances: (1) the operating system has access to both HSCs and (2) select buttons for both ports are depressed on the tape drive.

If one port fails, the operating system switches access to the operational port automatically, provided that the allocation class information has been defined correctly.

3.2.2 Dynamic Failover and Mount Verification

Dynamic failover occurs on dual-pathed tape drives if mount verification is unable to recover on the current path and an alternate path is available. The failover occurs automatically and transparently and then mount verification proceeds.

A device enters mount verification when an I/O request fails because the device has become inoperative. This might occur in the following instances:

- The device is placed offline accidentally.
- The active port of an HSC-connected drive fails.
- A hardware error occurs.
- The device is set to write protected during a write operation.

When the device comes back online, either through automatic failover or operator intervention, the operating system validates the volume, restores the tape to the position when the I/O failure occurred, and retries the failed request.

3.2.3 Tape Caching

The RV20, TA90, TK70, and TU81-Plus contain **write-back volatile** caches. The host enables write-back volatile caches explicitly, either on a per-unit basis or on a per-command basis. To enable caching on a per-unit basis, the user can enter the DCL MOUNT command specifying the qualifier /CACHE=TAPE_DATA.

The Backup utility enables caching on a per-command basis. The user can implement caching on a per-command basis at the QIO level by using the IO$M_NOWAIT function modifiers on commands where it is legal. (See Table 3–4.) In the unlikely event that cached data is lost, the system returns a fatal error and the device accepts no further I/O requests. The IO$M_FLUSH function code can be used to ensure that all write-back-cached data has been written out to the specified tape unit. The IO$_PACKACK, IO$_UNLOAD, IO$_REWINDOFF, and IO$_AVAILABLE function codes also flush the cache.

3.2.4 Master Adapters and Slave Formatters

The 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 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.

3.2.5 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$M_DATACHECK) on a read logical block, write logical block, read virtual block, write virtual block, read physical block, or write physical block I/O function.
- Per volume—You can specify the characteristics ‘‘data check all reads’’ and ‘‘data check all writes’’ when the volume is mounted. The *OpenVMS DCL Dictionary* describes volume mounting and dismounting. The *OpenVMS System Services Reference Manual* describes the Mount Volume ($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 *OpenVMS Record Management Services Reference 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 3.1.10 for information on TK50 data check.

Note

Read and write operations with data check can result in very slow performance on streaming tape drives.

3.2.6 Error Recovery

Error recovery 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$M_INHRETRY) inhibits all normal (nonspecial conditions) error recovery. If an error occurs, and the request includes that modifier, the operation is terminated immediately and the driver returns a failure status. IO$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.

3.2.7 Streaming Tape Systems

Streaming tape systems, such as the TK50, TK70, TU80, TU81, TU81-Plus, TA81, 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.
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 TU8*x* tapes and about 3 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 such as the 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.

3.3 Magnetic Tape Driver Device Information

You can obtain information on all magnetic tape device characteristics by using the Get Device/Volume Information ($GETDVI) system service. (See the *OpenVMS 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 3–1, 3–2, and 3–3 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 3–1 Magnetic Tape Device-Independent Characteristics

Characteristic[1]	Meaning
Dynamic Bits (Conditionally Set)	
DEV$M_AVL	Device is online and available.
DEV$M_FOR	Volume is foreign.
DEV$M_MNT	Volume is mounted.
DEV$M_RCK	Perform data check on all read operations.
DEV$M_WCK	Perform data check on all write operations.
Static Bits (Always Set)	
DEV$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.

[1]Defined by the $DEVDEF macro.

[2]This bit is located in DVI$_DEVCHAR2.

Table 3–2 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.

[1]Defined by the $MTDEF macro.

(continued on next page)

Table 3–2 (Cont.) Device-Dependent Information for Tape Devices

Characteristic[1]	Meaning	
MT$M_EOF	If set, a tape mark was encountered by the last operation to move tape.	
MT$M_BOT	If set, a beginning-of-tape (BOT) marker was encountered by the last operation to move the tape in the reverse direction.	
MT$M_PARITY	If set, all data transfers are performed with even parity. If clear (normal case), all data transfers are performed with odd parity. Only non-return-to-zero-inverted recording at 800 bits /inch can have even parity.	
MT$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 bits/inch
	MT$K_PE_1600	Phase-encoded recording, 1600 bits/inch
	MT$K_NRZI_800	Non-return-to-zero-inverted recording, 800 bits/inch
	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.

[1]Defined by the $MTDEF macro.

[2]Only for the TK50 and TZ30.

Table 3–3 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.

[1]Defined 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.

3.4 Magnetic Tape Function Codes

The 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.

3.4 Magnetic Tape Function Codes

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 3–4 lists the logical, virtual, and physical magnetic tape I/O functions and their function codes. These functions are described in more detail in the following paragraphs. Chapter 1 describes the QIO level interface to the magnetic tape device ACP.

Table 3–4 Magnetic Tape I/O Functions

Function Code	Arguments	Type[1]	Function Modifiers	Function
IO$_ACCESS	P1,[P2],[P3],[P4],[P5]	V	IO$M_CREATE IO$M_ACCESS	Search a tape for a specified file and access the file if found and IO$M_ ACCESS is set. If the file is not found and IO$M_CREATE is set, create a file at end-of-tape (EOT) marker.
IO$_ACPCONTROL	P1,[P2],[P3],[P4], [P5]	V	IO$M_DMOUNT	Perform miscellaneous control functions.[2]
IO$_AVAILABLE		P		Clear volume valid bit.
IO$_CREATE	P1,[P2][,[P3],[P4],[P5]	V	IO$M_CREATE IO$M_ACCESS	Create a file.
IO$_DEACCESS	P1,[P2],[P3],[P4],[P5]	V		Deaccess a file and, if the file has been written, write out trailer records.
IO$_DSE[3]		P	IO$M_NOWAIT	Erase a prescribed section of the tape.
IO$_FLUSH		L		Flush the controller cache to tape.
IO$_MODIFY	P1,[P2],[P3],[P4],[P5]	V		Write user labels.
IO$_PACKACK		P		Initialize volume valid bit.

[1] V = virtual; L = logical; P = physical.

[2] See Section 1.6.8 for additional information.

[3] Only for TMSCP drives, TZK50, and TZ30.

(continued on next page)

Table 3–4 (Cont.) Magnetic Tape I/O Functions

Function Code	Arguments	Type[1]	Function Modifiers	Function
IO$_READLBLK[10]	P1,P2	L	IO$M_DATACHECK[4] IO$M_INHRETRY IO$M_REVERSE[5]	Read logical block.
IO$_READPBLK[10]	P1,P2	P	IO$M_DATACHECK[4] IO$M_INHRETRY IO$M_REVERSE[5]	Read physical block.
IO$_READVBLK[10]	P1,P2	V	IO$M_DATACHECK[4] IO$M_INHRETRY IO$M_REVERSE[5]	Read virtual block.
IO$_REWIND		L	IO$M_INHRETRY IO$M_NOWAIT IO$M_RETENSION	Reposition tape to the beginning-of-tape (BOT) marker.
IO$_REWINDOFF		L	IO$M_INHRETRY IO$M_NOWAIT IO$M_RETENSION	Rewind and unload the tape on the selected drive.
IO$_SENSECHAR	[P1],[P2][6]	P	IO$M_INHRETRY	Sense the tape characteristics and return them in the I/O status block.
IO$_SENSEMODE	[P1],[P2][6]	L	IO$M_INHRETRY	Sense the tape characteristics and return them in the I/O status block.
IO$_SETCHAR	P1,[P2][6]	P		Set tape characteristics for subsequent operations.
IO$_SETMODE	P1,[P2][6]	L		Set tape characteristics for subsequent operations.
IO$_SKIPFILE	P1	L	IO$M_INHRETRY IO$M_NOWAIT[7]	Skip past a specified number of tape marks in either a forward or reverse direction.
IO$_SKIPRECORD	P1	L	IO$M_INHRETRY IO$M_NOWAIT[7]	Skip past a specified number of blocks in either a forward or reverse direction.
IO$_UNLOAD		L	IO$M_INHRETRY IO$M_NOWAIT	Rewind and unload the tape on the selected drive.

[1]V = virtual; L = logical; P = physical.

[4]Not for TS04 and TU80.

[5]Not for TUK50 and TQK50.

[6]The 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.

[7]Only for RV20, TA90, TK70, and TU81-Plus drives.

[10]On OpenVMS Alpha, P1 supports a 64-bit address.

(continued on next page)

Table 3–4 (Cont.) Magnetic Tape I/O Functions

Function Code	Arguments	Type[1]	Function Modifiers	Function
IO$_WRITELBLK[10]	P1,P2	L	IO$M_ERASE[8] IO$M_DATACHECK[4] IO$M_INHRETRY IO$M_INHEXTGAP[9] IO$M_NOWAIT[7]	Write logical block.
IO$_WRITEOF[10]		L	IO$M_INHRETRY IO$M_INHEXTGAP[9] IO$M_NOWAIT[7]	Write an extended interrecord gap followed by a tape mark.
IO$_WRITEPBLK[10]	P1,P2	P	IO$M_ERASE[8] IO$M_DATACHECK[4] IO$M_INHRETRY IO$M_INHEXTGAP[9] IO$M_NOWAIT[7]	Write physical block.
IO$_WRITEVBLK[10]	P1,P2	V	IO$M_DATACHECK[4] IO$M_INHRETRY IO$M_INHEXTGAP[9] IO$M_NOWAIT[7]	Write virtual block.

[1]V = virtual; L = logical; P = physical.

[4]Not for TS04 and TU80.

[7]Only for RV20, TA90, TK70, and TU81-Plus drives.

[8]Takes no arguments; valid only for TMSCP drives, TZK50, and TZ30.

[9]Only for TE16, TU45, and TU77.

[10]On OpenVMS Alpha, P1 supports a 64-bit address.

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.
- 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. On OpenVMS Alpha, P1 can be a 64-bit address.

- 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.)

Example 3–1 shows the correct method of defining the P1 parameter in an IO$_SKIPRECORD QIO.

Example 3–1 Defining the P1 Parameter in a IO$_SKIPRECORD QIO

```
   .
   .
   .
TAPE_CHAN:
        .WORD    0
IOSB:   .WORD    0
        .WORD    0
        .LONG    0
DEVICE: .ASCID   /$127$MUA0:/
RECORD: .LONG    2000
;
        .PSECT   CODE,EXE,NOWRT
;
        .ENTRY   MT_IO,^M<>
;
        $ASSIGN_S    CHAN=TAPE_CHAN,-
                 DEVNAM=DEVICE
        BLBC     R0,EXIT_ERROR
;
        $QIOW_S  CHAN=TAPE_CHAN,-
                 FUNC=#IO$_SKIPRECORD,-
                 IOSB=IOSB,-
                 P1=RECORD
        BLBC     R0,EXIT_ERROR
        $EXIT_S  R0
   .
   .
   .
EXIT_ERROR:
        $EXIT_S  R0
        .END     MT_IO
```

3.4.1 Read

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

The 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 3.4.

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.

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.

3.4.2 Write

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

The 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 3.4.

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.

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.

3.4.3 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.

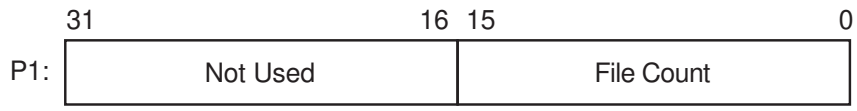
If the IO$M_RETENSION function modifier is specified and the device supports the retension operation, the rewind function positions the tape to the physical-end-of-tape (EOT) marker and rewinds the tape to the beginning-of-tape (BOT) marker. If the tape does not support the IO$M_RETENSION modifer, a SS$_ ILLIOFUNC error is returned.

IO$_REWIND has no function-dependent arguments.

3.4.4 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 3–1. 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 an unsigned number in the I/O status block.)

Figure 3–1 IO$_SKIPFILE Argument



ZK–0671–GE

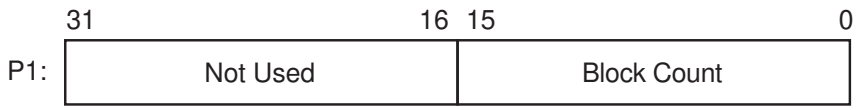
Only tape marks (when the tape moves in either direction) and the BOT marker (when the tape moves in reverse) are counted during a skip file operation. The BOT marker terminates a skip file function in the reverse direction. The end-of-tape (EOT) marker does not terminate a skip file function in either 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 3.4.5.1 describes this feature.

3.4.5 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 3–2. 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 an unsigned 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 3–2 IO$_SKIPRECORD Argument



ZK–0672–GE

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 3.4.5.1 describes this feature.

3.4.5.1 Logical End-of-Volume Detection

A skip file or skip record operation is terminated when 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).

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.

3.4.6 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.

3.4.7 Rewind Offline

The rewind offline function rewinds and unloads the tape on the selected drive.

The I/O operation is completed as soon as the tape movement is initiated. The actual finish of the mechanical rewind or unload operation may occur long after the I/O operation completes.

If the IO$M_RETENSION function modifier is specified and the device supports the retension operation, the rewind offline function positions the tape to the physical end-of-tape (EOT) marker and rewinds the tape to the beginning-of-tape (BOT) marker. If the tape does not support the IO$M_RETENSION modifer, a SS$_ILLIOFUNC error is returned.

No device or function-dependent arguments are used with IO$_REWINDOFF.

3.4.8 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.

3.4.9 Sense Tape Mode

The sense tape mode function senses the current device-dependent and extended device characteristics (see Tables 3–2 and 3–3).

The operating system provides the following function codes:

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

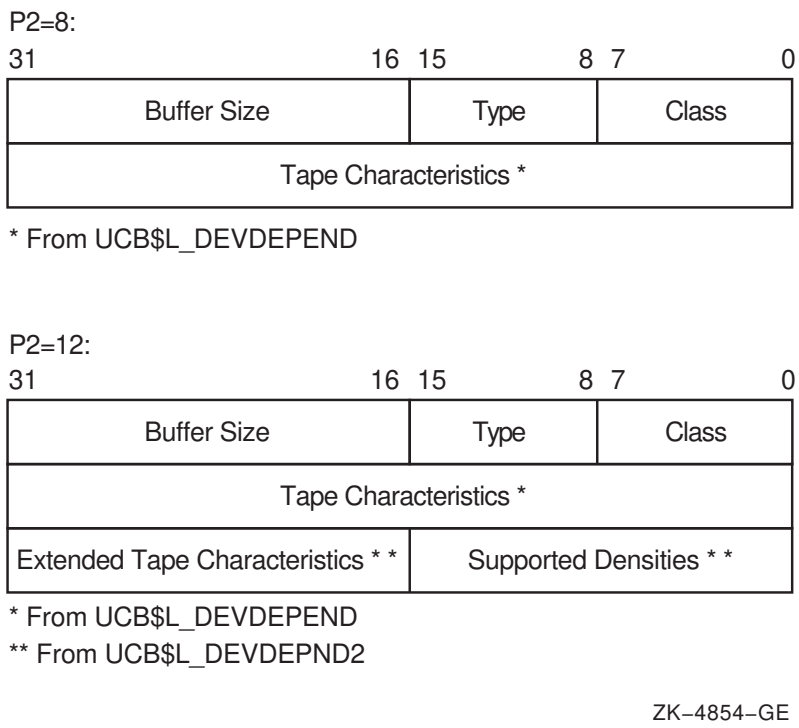
Sense mode requires logical I/O privilege. Sense characteristics requires physical I/O privilege. For TMSCP drives, the sense mode function returns magnetic tape information in a user-supplied buffer, which is specified by the following function-dependent arguments:

- P1—Optional. Address of a user-supplied buffer.
- P2—Optional. Length of a user-supplied buffer.

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 3–3). Figure 3–3 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 3–6). These characteristics are identical to the information returned by DVI$_DEVDEPEND (see Table 3–2 in Section 3.3).

Figure 3–3 Sense Mode P1 Buffer



3.4.10 Set Mode

Set mode operations affect the operation and characteristics of the associated magnetic tape device. The 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:

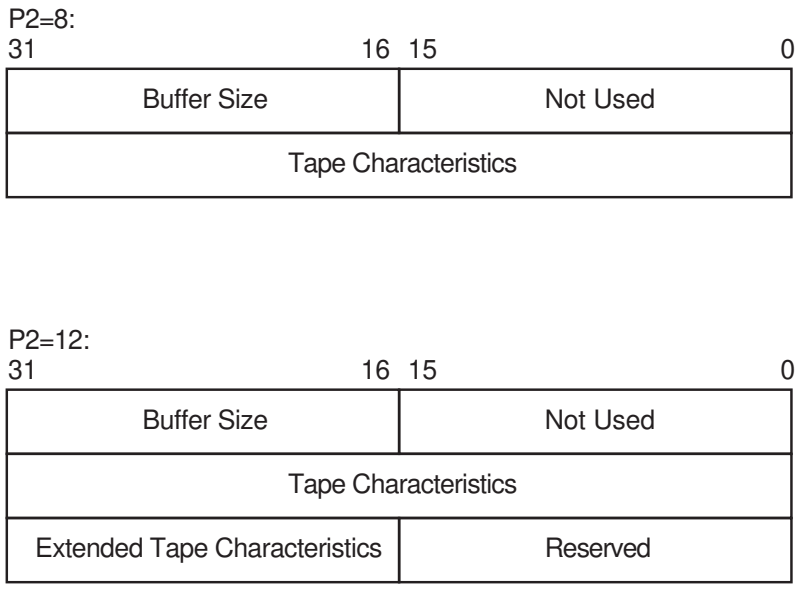
- IO$_SETMODE—Set mode
- IO$_SETCHAR—Set characteristics

These functions take the following device- or function-dependent arguments (other arguments are ignored):

- P1—The address of a characteristics buffer.
- P2—Optional. The length of the characteristics buffer. Default is 8 bytes. If a length of 12 bytes is specified, the third longword (which is for TMSCP drives only) specifies the extended tape characteristics.

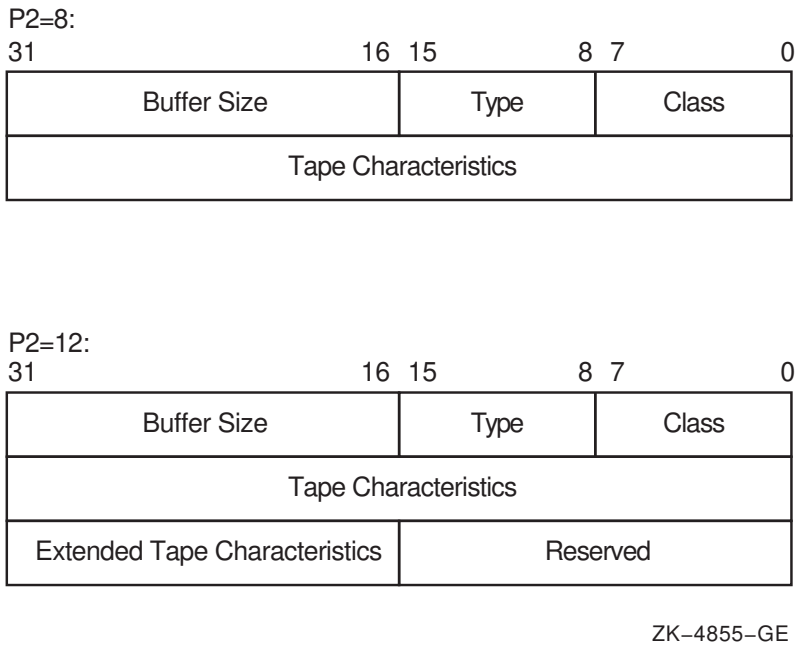
Figure 3–4 shows the P1 characteristics buffer for IO$_SETMODE. Figure 3–5 shows the same buffer for IO$_SETCHAR.

Figure 3–4 Set Mode Characteristics Buffer for IO$_SETMODE



ZK–4856–GE

Figure 3–5 Set Mode Characteristics Buffer for IO$_SETCHAR



ZK–4855–GE

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 3–5 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 3–6. The extended tape characteristics are defined by the $MT2DEF macro and are identical to the information returned by DVI$_DEVDEPEND2.

Table 3–5 Set Mode and Set Characteristics Magnetic Tape Characteristics

Characteristic[1]	Meaning	
MT$M_PARITY	If set, all data transfers are performed with even parity. If clear (normal case), all data transfers are performed with odd parity. Even parity can be selected only for non-return-to-zero-inverted recording at 800 bits/inch. Even parity cannot be selected for phase-encoded recording (tape density is MT$K_PE_1600) or group-coded recording (tape density is MT$K_GCR_6250) and is ignored.	
MT$V_DENSITY MT$S_DENSITY	Specifies the density at which all data transfers are performed. Tape density can be set only when the selected drive's tape position is at the BOT marker. Possible density values are as follows:	
	MT$K_DEFAULT	Default system density.
	MT$K_GCR_6250	Group-coded recording, 6250 bits /inch.
	MT$K_PE_1600	Phase-encoded recording, 1600 bits /inch.
	MT$K_NRZI_800	Non-return-to-zero-inverted recording, 800 bits/inch.
	MT$K_BLK_833	Cartridge block mode recording[2].
MT$V_FORMAT MT$S_FORMAT	Specifies the format in which all data transfers are performed. Possible format values are as follows:	
	MT$K_DEFAULT	Default system format.
	MT$K_NORMAL11	Normal PDP–11 format. Data bytes are recorded sequentially on tape with each byte occupying exactly one frame.

[1]Defined by the $MTDEF macro

[2]Only for the TK50 and TZ30

Table 3–6 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.

[1]Defined 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 3–2 in Section 3.6:

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.

3.4.11 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, and TZ30 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.

3.4.12 Modify

Specifying the ATR$C_USERLABEL or ATR$C_ENDLBLAST attributes with IO$_MODIFY results in a bad attribute error. If any other attributes are specified, the IO$_MODIFY function is treated as a no-operation; that is, the function returns success, but no action is performed.

3.4.13 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.

3.4.14 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 3.4.13). 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.

3.4.15 Flush

The flush function is used to ensure that all previously issued cached commands have fully completed. Normally, hosts use this function to establish or maintain synchronization with write-back cached commands issued to the specified tape unit. The I/O request does not complete until all cached data is written successfully to the media in the exact order that the user specified.

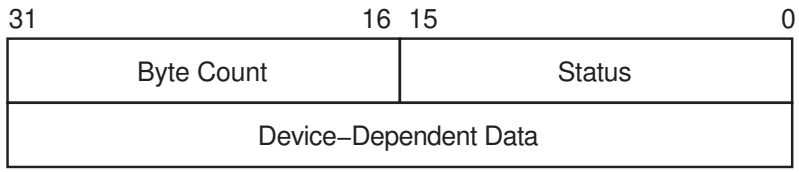
IO$_FLUSH

This function code takes no function-dependent arguments.

3.5 I/O Status Block

The I/O status block (IOSB) for QIO functions on magnetic tape devices is shown in Figure 3–6. Appendix A lists the status returns for these functions. (The OpenVMS system messages documentation provides explanations and suggested user actions for these returns.) Table 3–2 (in Section 3.3) lists the device-dependent data returned in the second longword. The IO$_SENSEMODE function can be used to return that data.

Figure 3–6 IOSB Contents



ZK–0675–GE

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 an unsigned number reflecting the number of blocks skipped, plus 1.

3.6 Magnetic Tape Driver Programming Examples

This section presents magnetic tape driver VAX MACRO programming examples.

Example 3–2 illustrates the recommended sequence for changing a device characteristic. It retrieves the current characteristics using a IO$_SENSEMODE request, sets the new characteristics bits, and then uses IO$_SETMODE to set the new characteristics.

Example 3–3 shows ways of specifying sense mode and set mode, both with and without a user buffer specified, and with user buffers of different lengths.

In addition, Example 3–4 shows how data is written to and read from magnetic tape, through the magnetic tape ACP.

Example 3–2 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)
        .
        .
        .
```

Example 3–3 Set Mode and Sense Mode Program Example

```
        .PSECT        IMPURE, NOEXE, NOSHR

        $IODEF

DEVICE_NAME:                            ; Name of device
        .ASCID        /MUA0/            ;

CHANNEL:                                ; 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
```

(continued on next page)

Example 3–3 (Cont.) Set Mode and Sense Mode Program Example

```
$QIOW_S -                               ; Get current characteristics
        FUNC        = #IO$_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

$QIOW_S -                               ; Set new characteristics
        FUNC        = #IO$_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
```

(continued on next page)

Example 3–3 (Cont.) Set Mode and Sense Mode Program Example

```
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
```

Example 3–4 MAGNETIC_TAPE.MAR Device Characteristic 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.
;
```

(continued on next page)

Example 3–4 (Cont.) MAGNETIC_TAPE.MAR Device Characteristic Program Example

```
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.
;
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.
;
```

(continued on next page)

Example 3–4 (Cont.) MAGNETIC_TAPE.MAR Device Characteristic Program Example

```
        $ASSIGN_S TAPENAME,TAPECHAN     ;Assign tape unit
        CMPW    #SS$_NORMAL,R0          ;Success?
        BSBW    ERRCHECK                ;Find out

;
; Now create and access the file MYDATA.DAT.
;
        $QIOW_S CHAN=TAPECHAN,-         ;Channel is magtape
                FUNC=#IO$_CREATE!IO$M_ACCESS!IO$M_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,R0          ;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,R0          ;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
```

(continued on next page)

Example 3–4 (Cont.) MAGNETIC_TAPE.MAR Device Characteristic Program Example

```
;
; 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.
;
        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,R0          ;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$M_REVERSE,- ;Function is read
                -                       ;reverse
                IOSB=IOSTATUS,-         ;Define I/O status quadword
                P1=BUFFER,-             ;And buffer address
                P2=R3                   ;R3 bytes
        CMPW    #SS$_NORMAL,R0          ;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.
;
```

(continued on next page)

Example 3–4 (Cont.) MAGNETIC_TAPE.MAR Device Characteristic Program Example

```
        $QIOW_S CHAN=TAPECHAN,-         ;Channel is magtape
                FUNC=#IO$_DEACCESS,-    ;Deaccess function
                P1=FIB_DESCR,-          ;File information block (required)
                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.
;
MISMATCH:                               ;
        BRB     EXIT                    ;Exit

ERRCHECK:                               ;
        BNEQ    EXIT                    ;If not success, exit
        RSB                             ;Otherwise, return

        .END    MAGTAPE_EXAMPLE
```

4
Mailbox Driver

The 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 way to perform read and write operations between processes.

The *OpenVMS Programming Concepts Manual* and the *OpenVMS System Services Reference Manual* contain additional information on the use of mailboxes.

4.1 Mailbox Operations

This section describes the following mailbox operations:

- Creating mailboxes
- Deleting mailboxes
- Protecting mailboxes

4.1.1 Creating Mailboxes

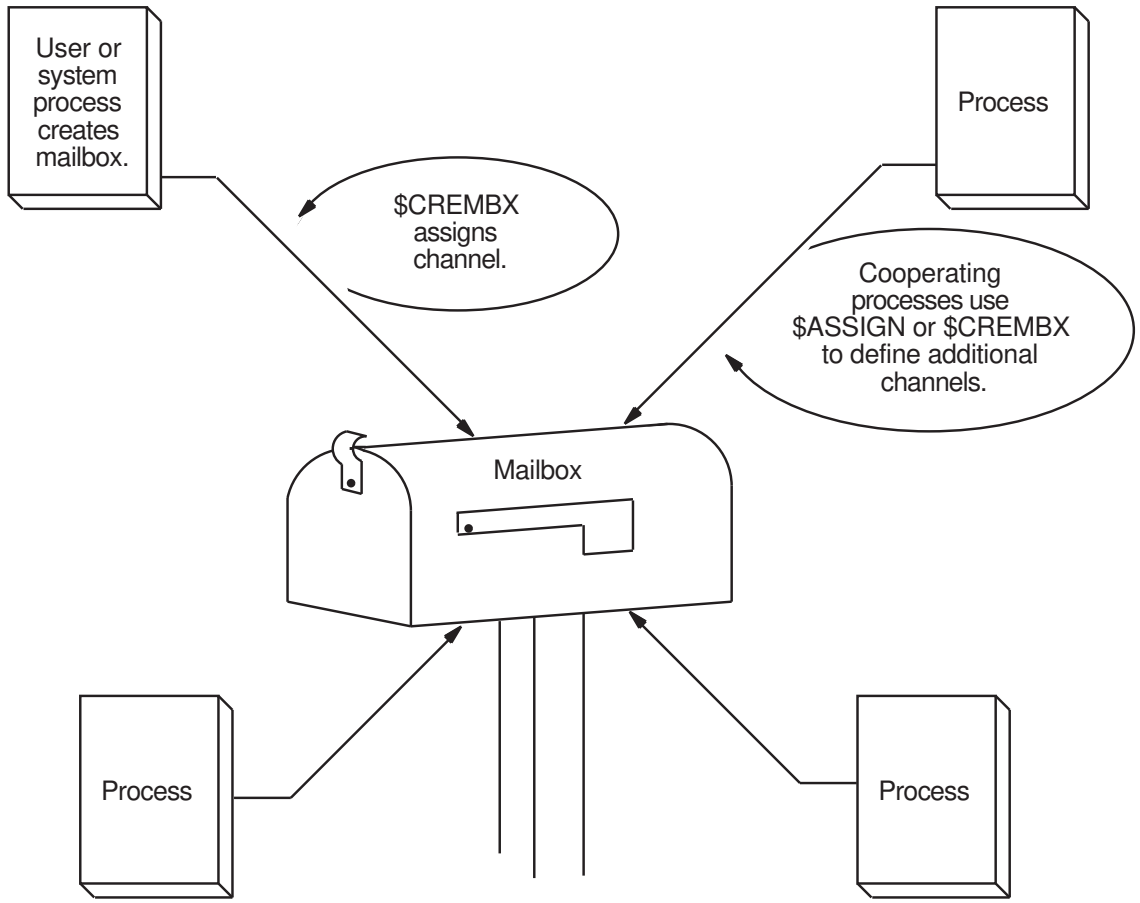
To create a mailbox and assign a channel and logical name to it, a process uses the Create Mailbox and Assign Channel ($CREMBX) system service. A logical name can optionally be associated with the mailbox. If a logical name is specified for the mailbox, the system enters the logical name in a logical name table and gives it an equivalence name of MBA*n*, where *n* is a unique unit number.

$CREMBX also establishes the characteristics of the mailbox. These characteristics include a protection mask, a permanence indicator, maximum message size, buffer quota, and direction in which I/O can be performed (read, write, or read/write). A mailbox is created as either temporary or permanent; both types require privilege to create. Applications and restrictions on use of temporary and permanent mailboxes are described in the sections that follow. (See the *OpenVMS System Services Reference Manual* for additional information on creating mailboxes.)

Other processes can assign additional channels to a mailbox using either the $CREMBX or the Assign I/O Channel ($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. Channels assigned to the mailbox can specify the direction that I/O can be performed on the channel.

Figure 4–1 illustrates the use of $CREMBX and $ASSIGN.

Figure 4–1 Multiple Mailbox Channels



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 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 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).

Channels can be assigned to mailboxes as bidirectional (read/write), read only, or write only. This allows for greater synchronization between users of the mailbox. To specify a unidirectional channel to the mailbox, specify the **flags** argument for the $CREMBX or $ASSIGN system services.

The **flags** argument is a longword bit mask that enables you to specify that the channel assigned to the mailbox is a read-only or write-only channel. If the **flags** argument is not specified, the default channel behavior is read/write. A channel assigned to the mailbox as read only is considered a reader. A channel assigned to the mailbox as write only is considered a writer. A channel assigned to the mailbox as read/write is considered both a reader and a writer.

For the $ASSIGN system service, the $AGNDEF macro defines a symbolic name for each flag bit. These flags are as follows:

- AGN$M_READONLY—When this flag is specified, $ASSIGN assigns a read-only channel to the mailbox device. An attempt to issue a $QIO WRITE operation on the mailbox channel causes an illegal I/O operation error.
- AGN$M_WRITEONLY—When this flag is specified, $ASSIGN assigns a write only channel to the mailbox device. An attempt to issue a $QIO READ operation on the mailbox channel causes an illegal I/O operation error.

For the $CREMBX system service, the $CMBDEF macro defines a symbolic name for each flag bit. These flags are as follows:

- CMB$M_READONLY—When this flag is specified, $CREMBX assigns a read-only channel to the mailbox device. An attempt to issue a $QIO WRITE operation on the mailbox channel causes an illegal I/O operation error.
- CMB$M_WRITEONLY—When this flag is specified, $CREMBX assigns a write-only channel to the mailbox device. An attempt to issue a $QIO READ operation on the mailbox channel causes an illegal I/O operation error.

See the *OpenVMS System Services Reference Manual* for a syntax description of the $CREMBX and $ASSIGN system services.

The programming examples at the end of this section (Section 4.5) illustrate mailbox creation, interprocess communication, and synchronization.

4.1.2 Deleting Mailboxes

As each process finishes using a mailbox, it deassigns the channel using the Deassign I/O Channel ($DASSGN) system service. Temporary mailboxes or permanent mailboxes that have been marked for deletion are actually deleted when no more channels are assigned to them.

If a mailbox channel is deassigned, any incomplete I/O requests on the mailbox channel for the process deassigning the channel are removed.

Permanent mailboxes that have not been marked for deletion must be explicitly deleted using the Delete Mailbox ($DELMBX) system service. An explicit deletion can occur at any time. As is true for temporary mailboxes, the mailbox is deleted when no processes have channels assigned to it.

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.)

4.1.3 Mailbox Message Format

There is no standardized format for mailbox messages and none is imposed on users.

4.1.4 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 4.3.6).

For additional information on temporary mailboxes and mailbox protection, see the description of the $CREMBX system service in the *OpenVMS System Services Reference Manual*.

4.2 Mailbox Driver Device Information

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

$GETDVI returns mailbox characteristics when you specify the item code DVI$_ DEVCHAR. Table 4–1 lists these characteristics, which are defined by the $DEVDEF macro.

Table 4–1 Mailbox Characteristics

Characteristic[1]	Meaning
Dynamic Bits (Conditionally Set)	
DEV$M_SHR	Device is shareable.
DEV$M_AVL	Device is available.
Static Bits (Always Set)	
DEV$M_REC	Device is record-oriented.
DEV$M_IDV	Device is capable of input.
DEV$M_ODV	Device is capable of output.
DEV$M_MBX	Device is a mailbox.

[1]Defined 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.

4.3 Mailbox Function Codes

The mailbox I/O functions are read, write, write end-of-file, set attention AST, wait for writer/reader, and get mailbox information.

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.

4.3.1 Read

Read mailbox functions are used to obtain messages written to the mailbox. The operating system provides the following mailbox function codes:

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

IO$_READLBLK, IO$_READVBLK, and IO$_READPBLK all perform the same operation. To issue a read request, a process can specify any of the read function codes.

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. If P2 specifies a zero-length buffer, P1 is ignored. On OpenVSM Alpha, P1 can be a 64-bit address.
- 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 modifiers can be specified with a read request:

- IO$M_WRITERCHECK—Complete the I/O operation with SS$_NOWRITER status if the mailbox is empty and no write channels are assigned to the mailbox. If no writer is assigned to the mailbox when the $QIO is issued and no data is in the mailbox, the $QIO completes immediately. If no data is in the mailbox but a writer is assigned, the $QIO operation completes when either a message is written or all writers deassign their channels to the mailbox. IO$M_WRITERCHECK is ignored if the channel on which it is issued is read/write because a writer is always assigned.
- IO$M_NOW—Complete the I/O operation immediately with no wait for a write request from another process
- IO$M_STREAM—Ignore QIO record boundaries. The read operation transfers message data to the user's buffer until either P2 bytes are transferred, all message data currently in the mailbox is transferred, or an end-of-file message is encountered. If a WRITEOF message is within the records required to be read in order to fulfill the request for P2 bytes, the

read request terminates successfully with the bytes it was able to read before finding the WRITEOF message and the end-of-file message becomes the first message in the mailbox. The next read request processes the end-of-file message. If the read request is a READ STREAM, then the request must be for greater than 0 bytes. $QIO READ STREAM can return fewer than P2 bytes with a return value of SS$_NORMAL if the mailbox is emptied by the $QIO READ STREAM request or a WRITEOF message is encountered.

Figure 4–2 illustrates $QIO READ STREAM operations.

A READ IO$M_STREAM (without IO$M_NOW specified) on an empty mailbox waits until some data has been written to the mailbox. It terminates with:

- 0 bytes read if the next data written is an end-of-file message.
- Fewer than P2 bytes read if the next data written is less than P2 bytes but greater than 0 bytes. (READ IO$M_STREAM ignores writes of 0 bytes.)
- P2 bytes read if the next data written is greater than or equal to P2 bytes.

If a $QIO READ STREAM is fulfilled by multiple $QIO WRITE requests, the sender PID returned in the IOSB of the $QIO READ STREAM reflects the first write request. A $QIO READ STREAM is charged BUFQUO for the request. This BUFQUO is released when the read request is met. A $QIO READ STREAM request that would cause BUFQUO to be exceeded for the mailbox when the mailbox has no writes pending returns an SS$_EXQUOTA error.

A $QIO READ STREAM issued to a mailbox that would cause BUFQUO to be exceeded because BUFQUO is occupied by write requests still executes. This happens because by allowing the mailbox to temporarily exceed BUFQUO, BUFQUO is freed. Similarly, a $QIO WRITE that is issued to a mailbox that would cause BUFQUO to be exceeded, because the BUFQUO is occupied by read stream requests, still executes.

Reads of 0-bytes are handled differently depending on which functional modifiers are specified. If IO$M_STREAM is specified, then the $QIO returns SS$_NORMAL with 0 bytes read. The contents of the mailbox remain exactly as they were before the $QIO was issued. A $QIO READ STREAM of 0 bytes does not remove a 0 byte record, nor does it remove an end-of-file marker. If IO$M_STREAM is not specified, then $QIO returns one of the following:

- SS$_NORMAL (if 0 bytes were written with the corresponding $QIO WRITE performed)
- SS$_BUFFEROVF (if the corresponding $QIO WRITE wrote more than 0 bytes with 0 bytes read)
- SS$_ENDOFFILE (if a WRITEOF function was performed as the corresponding $QIO write function)

For a 0-byte nonstream read, a record is actually removed from the mailbox in order to meet the $QIO READ request. Note that the use of the word ‘‘immediately’’ does not imply that synchronization of the $QIO request should not be performed.

Figure 4–2 $QIO READ STREAM Operation

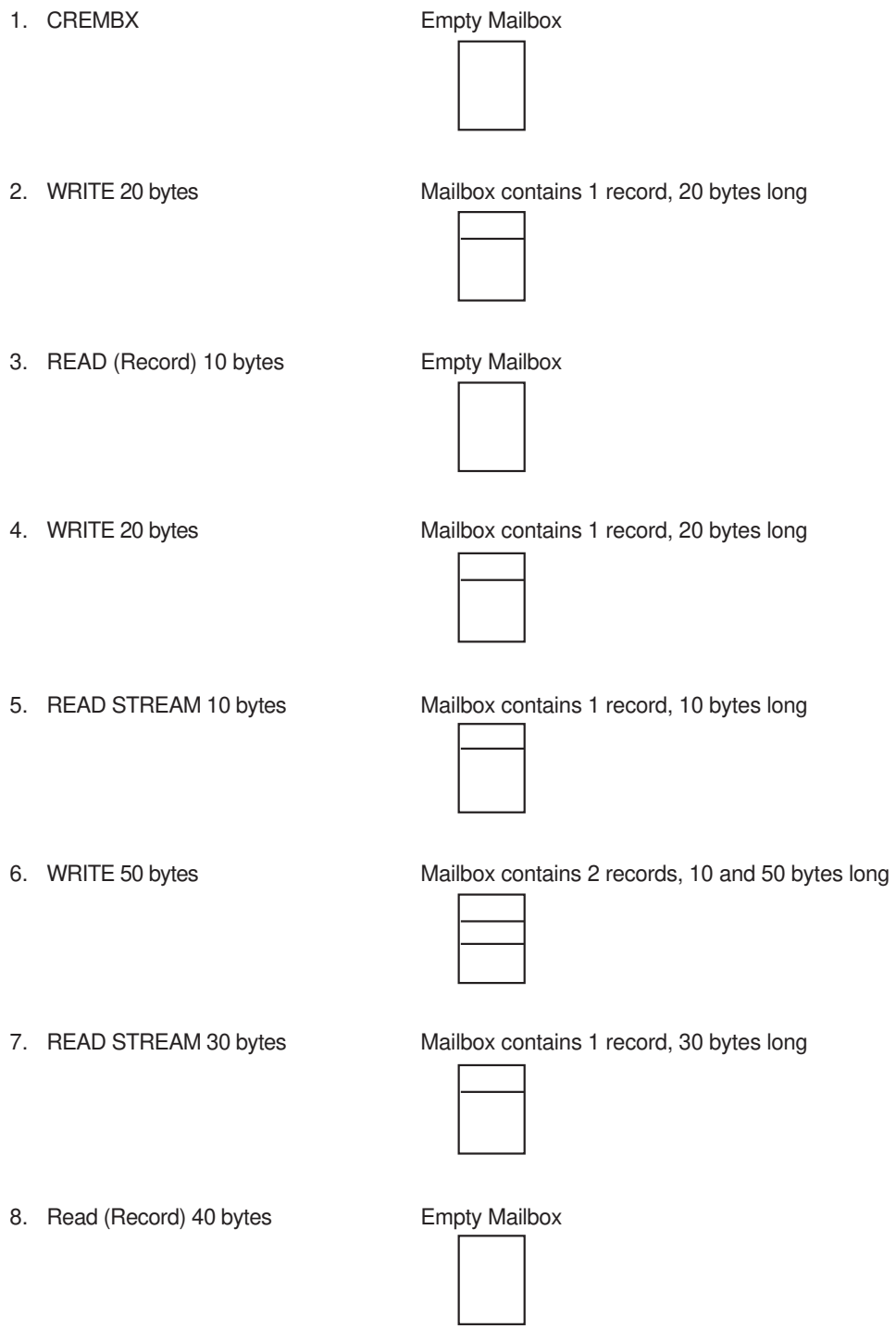
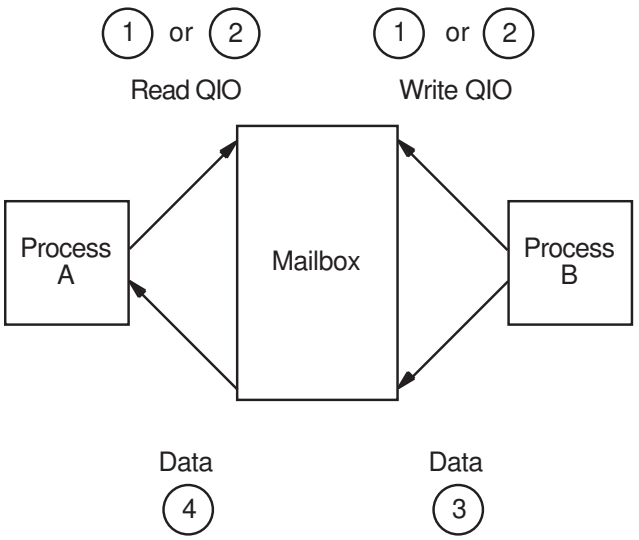


Figure 4–3 illustrates the read mailbox functions. In this figure, process A reads a mailbox message written by process B. As the figure indicates, a mailbox read request requires a corresponding mailbox write request (except in the case of an error). The requests can be made in any sequence; the read request can either precede or follow the write request.

Figure 4–3 Read Mailbox



Note: Numbers indicate order of events.

ZK–0679–GE

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 process B's write 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$M_NOW function modifier, the read operation is completed immediately. That is, no data is transferred from process B to process A, and process A's request is not queued until process B issues the write request. In this case, the I/O status returned to process A is SS$_ENDOFFILE.

If process B sends a message (with no function modifier; see Section 4.3.2) 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, regardless of whther IO$M_NOW is specified on the read request.

4.3.2 Write

Write mailbox functions are used to transfer data from a process to a mailbox. The operating system provides the following mailbox function codes:

- IO$_WRITEVBLK—Write virtual block
- IO$_WRITELBLK—Write logical block
- IO$_WRITEPBLK—Write physical block

IO$_WRITEVBLK, IO$_WRITELBLK, and IO$_WRITEPBLK all perform the same operation. To issue a write request, a process can specify any of the write function codes.

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. On OpenVMS Alpha, P1 can be a 64-bit address.
- 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$M_READERCHECK—Completes the I/O operation immediately, with SS$_NOREADER status, if no read channels are assigned to the mailbox. If a $QIO WRITE with IO$M_READERCHECK is issued and is outstanding and all read channels assigned to the mailbox are then deassigned, the $QIO completes with SS$_NOREADER status. IO$M_READERCHECK is ignored if the channel on which it is issued is bidirectional read/write, because there is always a reader assigned. If SS$_NOREADER is returned for a write request, the $QIO WRITE operation does not place any data in the mailbox. If SS$_NOREADER is returned for a write end-fo-file message request, the $QIO WRITE operation does not place the end-of-file marker in the mailbox.
- IO$M_NOW—Complete the I/O operation immediately without waiting for another process to read the mailbox message. $QIO WRITE, without IO$M_ NOW specified, does not complete until the data is read. $QIO WRITE NOW completes when the data is in the mailbox. If both IO$M_READERCHECK and IO$M_NOW are specified and no read channel is assigned to the mailbox, a status of SS$_NOREADER is returned and the data is not placed in the mailbox. If a read channel is assigned, the IO$M_READERCHECK modifier is ignored.
- IO$M_NORSWAIT—If the mailbox is full, the I/O operation fails with a status return of SS$_MBFULL rather than placing the process in resource wait mode. Note that IO$M_NORSWAIT does not disable resource waits that may occur elsewhere in the $QIO operation. For example, IO$M_NORSWAIT does not affect any resource waiting that occurs when I/O processing routines try to allocate an I/O request packet while passing the I/O request to the mailbox driver.

A $QIO WRITE of 0 bytes causes a 0-byte long message to be placed in the mailbox. When this data is read by a $QIO READ without IO$M_STREAM specified, the $QIO READ returns an SS$_NORMAL status and 0 bytes. When this data is read by a $QIO READ STREAM in an attempt to read P2 bytes (P2 being greater than 0), the data is ignored. However, a $QIO READ STREAM of 0 bytes has no effect on the mailbox. A $QIO WRITE READERCHECK of 0 bytes,

when no read channel is assigned to the mailbox, returns an SS$_NOREADER error and the 0-byte record is not placed in the mailbox. A message that is 0 bytes long is charged 1 byte of mailbox BUFQUO.

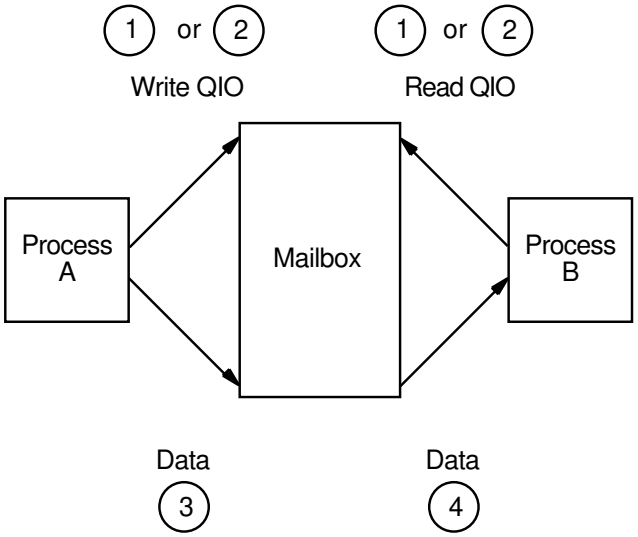
Figure 4–4 illustrates the write mailbox function. In this figure, process A writes a message to be read by process B. As in the read request example, a mailbox write request requires a corresponding mailbox read request (unless an error occurs) and the requests can be made in any sequence.

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.

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.

Figure 4–4 Write Mailbox



Note: Numbers indicate order of events.

ZK–0680–GE

4.3.3 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. The message count of the Get Mailbox Information function reflects this end-of-file message; however, the mailbox byte count of this function does not include end-of-file markers. An end-of-file message is charged 1 byte of mailbox BUFQUO.

This function takes no arguments. The operating system provides the following function code:

IO$_WRITEOF—Write end-of-file message

The following function modifiers can be specified with a write end-of-file request:

- IO$M_READERCHECK—Completes the I/O operation immediately, with SS$_NOREADER status, if no read channels are assigned to the mailbox. If a $QIO WRITEOF with IO$M_READERCHECK is issued and is outstanding and all read channels assigned to the mailbox are then deassigned, the $QIO completes with SS$_NOREADER status. IO$M_READERCHECK is ignored if the channel on which it is issued is bidirectional read/write, because there is always a reader assigned. If SS$_NOREADER is returned for a write end-of-file message request, the $QIO WRITEOF operation does not place the end-of-file marker in the mailbox.
- IO$M_NOW—Complete the I/O operation immediately without waiting for another process to read the mailbox message. If both IO$M_READERCHECK and IO$M_NOW are specified, and no read channel is assigned to the mailbox, a status of SS$_NOREADER is returned and the end-of-file message is not placed in the mailbox.
- IO$M_NORSWAIT—If the mailbox is full, the I/O operation fails with a status return of SS$_MBFULL instead of placing the process in resource wait mode. Note that IO$M_NORSWAIT does not disable resource waits that may occur elsewhere in the $QIO operation. For example, IO$M_NORSWAIT does not affect any resource waiting that occurs when I/O processing routines try to allocate an I/O request packet while passing the I/O request to the mailbox driver.

4.3.4 Set Attention AST

Set attention AST functions specify that an AST be delivered to the requesting process in the following cases:

- When a cooperating process places a read request for which no write request is pending in a designated mailbox. This is called an unsolicited read request.
- When a cooperating process places a write request for which no read request is pending in a designated mailbox. This is called an unsolicited write request.
- When room becomes available in the 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 write request 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 operating system provides the following function codes:

- IO$_SETMODE!IO$M_READATTN—Read attention AST
- IO$_SETMODE!IO$M_WRTATTN—Write attention AST
- IO$_SETMODE!IO$M_MB_ROOM_NOTIFY—Room in the mailbox attention AST

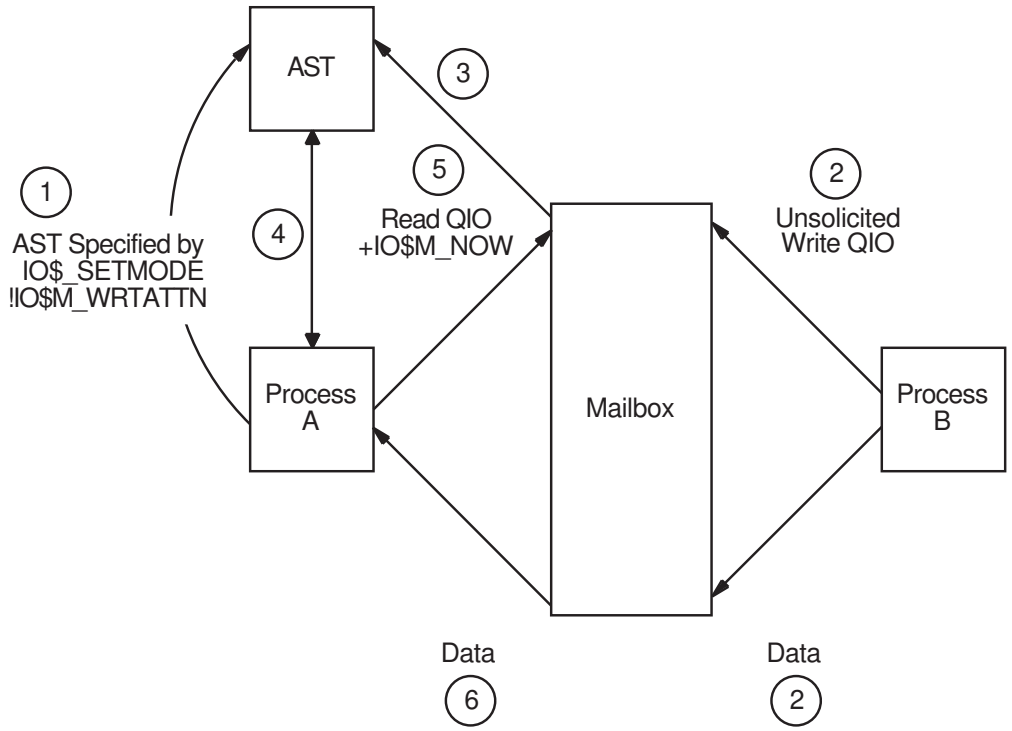
These function codes take the following device- or function-dependent arguments:

- 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 repeat notification. All 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 4–5 illustrates the write attention AST function. In this figure, an AST is set to notify process A when process B sends an unsolicited message.

Figure 4–5 Write Attention AST (Read Unsolicited Data)



Note: Numbers indicate order of events.

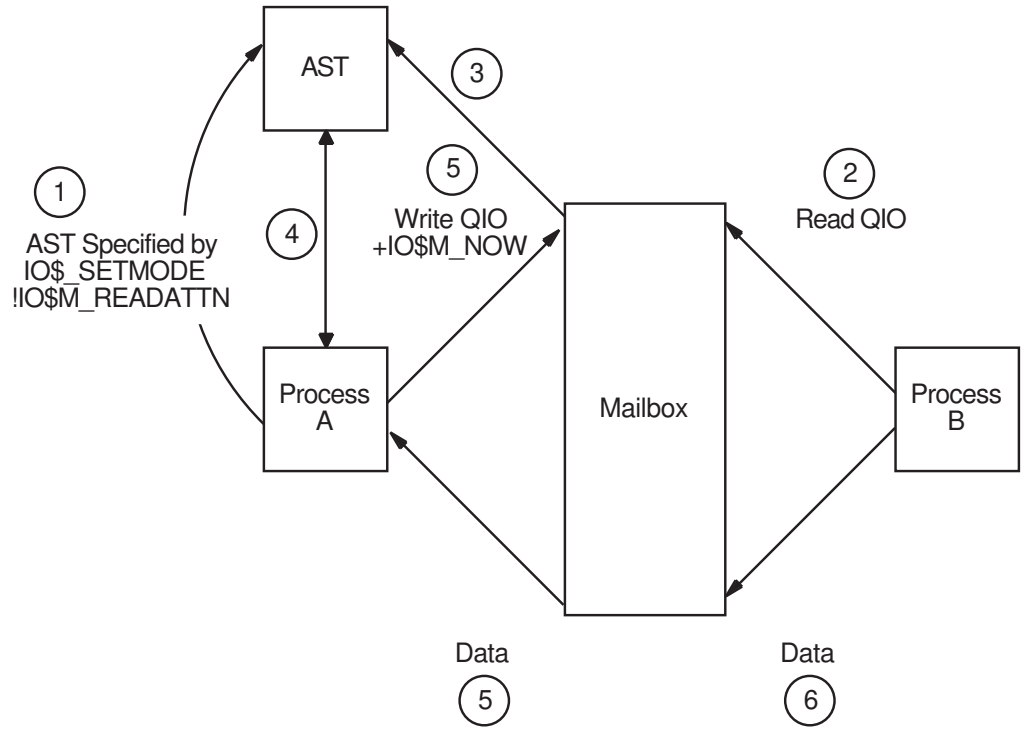
ZK–0681–GE

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 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 4–6 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.

Figure 4–6 Read Attention AST



Note: Numbers indicate order of events.

ZK–0682–GE

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

If several requesting processes 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.

4.3.5 Wait for Writer/Reader

The wait for writer/reader mailbox driver function waits until a channel is assigned to the mailbox with the requested access direction. This function returns immediately if a channel is already assigned to the mailbox with the proper access direction. This function always returns immediately if issued on a bidirectional mailbox channel. Any channel assigned bidirectionally to the mailbox satisfies both types of wait requests.

The wait function requires the same synchronization techniques as all other $QIO functions. $QIO Wait should not be issued without any synchronization of its completion. If no synchronization is performed, the program behaves as if no $QIO Wait function had been issued (except for the small delay caused by issuing the $QIO Wait).

The following function codes and modifiers are provided:

- IO$_SETMODE!IO$M_READERWAIT—Wait for a read channel to be assigned to the mailbox.
- IO$_SETMODE!IO$M_WRITERWAIT—Wait for a write channel to be assigned to the mailbox.

These function codes require no function-dependent arguments.

These functions are enabled only once. Once the $QIO operation completes, these functions must be explicitly reenabled.

4.3.6 Set Protection

On VAX and Alpha systems, the set protection functions allow the user to set volume protection on a mailbox (see Section 4.1.4). The requester must either be the owner of the mailbox or have BYPASS privilege. The OpenVMS 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:

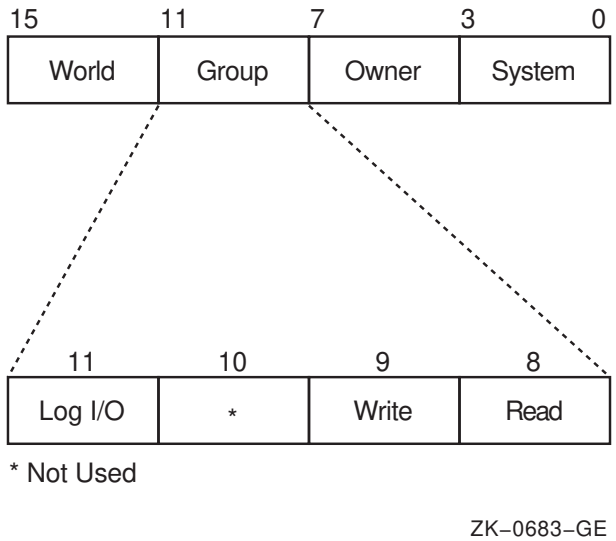
P2—A volume protection mask

The protection mask specified by P2 is a 16-bit mask with 4 bits for each class of owner: SYSTEM, OWNER, GROUP, and WORLD, as shown in Figure 4–7.

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 4–10) 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.

Figure 4–7 Protection Mask



4.3.7 Get Mailbox Information

The get mailbox information function allows the user to find out the number of unread messages and bytes in the mailbox. The following function code is provided:

IO$_SENSEMODE—Get mailbox contents information

The following function codes and modifiers are provided:

- IO$_SENSEMODE!IO$M_READERCHECK—If a $QIO SENSEMODE with IO$M_READERCHECK is issued and no read channels are assigned to the mailbox, then the SS$_NOREADER condition value is returned in the IOSB.
- IO$_SENSEMODE!IO$M_WRITERCHECK—If a $QIO SENSEMODE with IO$M_WRITERCHECK is issued and no write channels are assigned to the mailbox, then the SS$_NOWRITER condition value is returned in the IOSB.

These function codes require no function-dependent arguments.

4.4 I/O Status Block

On VAX and Alpha systems, the I/O status blocks (IOSB) for mailbox read, write, set protection, and get mailbox information QIO functions are shown in Figures 4–8, 4–9, 4–10, and 4–11.

Appendix A lists the I/O status returns for these functions. In addition to the IOSB return values, the following statuses can be returned in R0 by the call to the system service:

SS$_ACCVIO
SS$_EXQUOTA
SS$_ILLIOFUNC
SS$_INSFMEM
SS$_MBFULL
SS$_MBTOOSML
SS$_NOPRIV
SS$_NORMAL

(The OpenVMS system messages documentation provides explanations and suggested user actions for both types of returns.)

Figure 4–8 IOSB Contents—Read Function

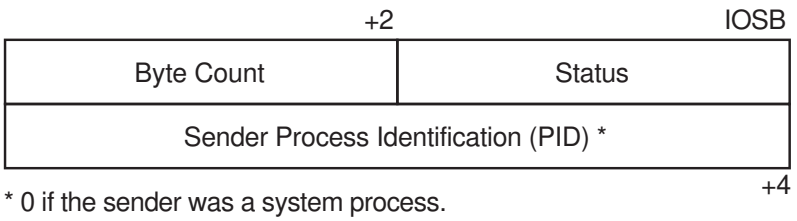


Figure 4–9 IOSB Contents—Write Function

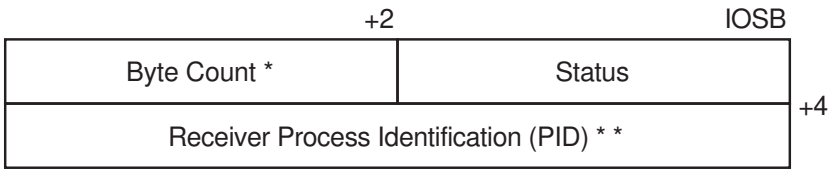


Figure 4–10 IOSB Contents—Set Protection Function

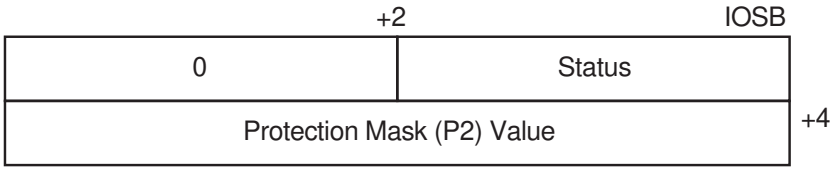
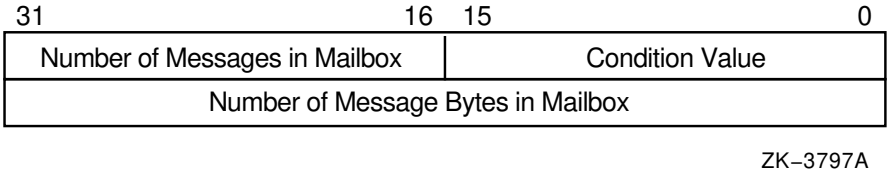


Figure 4–11 IOSB Contents—Get Mailbox Information Function



4.5 Mailbox Driver Programming Examples

This section contains the following programming examples:

- Example 4–1 shows a VAX MACRO program that creates a mailbox and puts mail into it.
- Example 4–2 assigns a read-only channel to the mailbox.
- Example 4–3 assigns a write-only channel to the mailbox.

Example 4–1 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.

Example 4–2 and Example 4–3 work together from two separate processes and illustrate the unidirectional mailbox synchronization features. Note that these examples require the /STANDARD=VACC qualifier when compiled on Alpha systems.

Example 4–2 performs the following functions:

1. Assigns a read-only channel to the mailbox.
2. Waits for another program to assign a writable channel to the mailbox.
3. Reads, using the IO$M_WRITERCHECK function modifier, what has been written to the mailbox.
4. When SS$_NOWRITER is returned from the read operation, goes back to Step 2 and waits for another writer.

Example 4–3 is a writer to the mailbox. It performs the following functions:

1. Assigns a write-only channel to the mailbox.
2. Waits for a reader.
3. Gathers user input until the user enters Ctrl/Z, then writes that input to the mailbox.

Example 4–1 Mailbox Driver Program Example 1

```
; ****************************************************************************
;
        .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

;
; Finally, allocate space for the I/O status quadword.
;
```

(continued on next page)

Example 4–1 (Cont.) Mailbox Driver Program Example 1

```
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,R0          ;Successful mailbox creation?
        BSBW    ERROR_CHECK             ;Find out
        $ASSIGN_S -                     ;Assign channel
                DEVNAM=DEVICE_DESCR,-   ;Device descriptor
                CHAN=DEVICE_CHANNEL     ;Channel
        CMPW    #SS$_NORMAL,R0          ;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,R0          ;Successful write request?
        BSBW    ERROR_CHECK             ;Find out

;
; Read the mailbox.
;
        $QIOW_S FUNC=#IO$_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,R0          ;Successful read request?
        BSBW    ERROR_CHECK             ;Find out
```

(continued on next page)

Example 4–1 (Cont.) Mailbox Driver Program Example 1

```
;
; 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
```

Example 4–2 assigns a read only channel to the mailbox.

Example 4–2 Mailbox Driver Program Example 2

```
/*
 * C program to demonstrate the new features of the Mailbox driver
 * This program is a Mailbox READER.  It assigns a READ_ONLY channel to the
 * mailbox.  It's partner program is a Mailbox WRITER.
 */

#include "sys$library:descrip"          /* Descriptor structure definitions  */
#include "sys$library:libdef"           /* LIB RTL symbol definitions        */
#include "sys$library:stdio"            /* Standard C symbol definitions     */
#include "decc$library:[include]ssdef"  /* System Service status code def'ns */
#include "decc$library:[include]cmbdef" /* CREMBX definitions                */
#include "decc$library:[include]agndef" /* ASSIGN definitions                */
#include "decc$library:[include]iodef"  /* I/O definitions                   */

#define $ARRAY_DESCRIPTOR(name,size,array_name)                   \
    char array_name[ size ];                                      \
    struct dsc$descriptor_s name =                                \
       { size, DSC$K_DTYPE_T, DSC$K_CLASS_S, array_name }

main()
```

(continued on next page)

Example 4–2 (Cont.) Mailbox Driver Program Example 2

```
{
# define max_msg_len 256
$DESCRIPTOR(mailbox_name_desc,"MAILBOX_EXAMPLE");
$ARRAY_DESCRIPTOR(read_buffer_desc,max_msg_len,read_buffer);
int status, mailbox_channel;
int true=1, false = 0;
struct io_status_block {                        /*  I/O status block */
    short int condition;
    short int count;
    int dev;
    } iosb;

/*
 * Create a permanent mailbox with a READONLY channel.  It's logical name
 *  will be entered into the LNM$PERMANENT_MAILBOX logical name table.
 */

    SYS$CREMBX(1,&mailbox_channel,0,0,0,0,&mailbox_name_desc,CMB$M_READONLY);

/*
 * Mark it for deletion
 */
    SYS$DELMBX(mailbox_channel);

/*
 * Loop forever, first waiting til a WRITE channel is assigned to the mailbox
 * and then reading data from it until the WRITER deassigns.
 */
    while (TRUE)
    {
       /* First, check to see if there is a WRITER assigned to the mailbox */
        status = SYS$QIOW (
            0,
            mailbox_channel,
            IO$_SENSEMODE|IO$M_WRITERCHECK,
            &iosb,
            0,0,
            0,0,0,0,0,0);

        /* If there was no WRITER, then wait for one.*/
        if (iosb.condition == SS$_NOWRITER)
        status = SYS$QIOW (
            0,
            mailbox_channel,
            IO$_SETMODE|IO$M_WRITERWAIT,
            &iosb,
            0,0,
            0,0,0,0,0,0);
```

(continued on next page)

Example 4–2 (Cont.) Mailbox Driver Program Example 2

```
        /*
         * While the status is good, READ from the mailbox, and write it to
         */
        while (status == SS$_NORMAL)
        {
            status = SYS$QIOW(
                0,
                mailbox_channel,
                IO$_READVBLK|IO$M_WRITERCHECK,
                &iosb,
                0,0,
                read_buffer_desc.dsc$a_pointer,max_msg_len,
                0,0,0,0);
            status=iosb.condition;
            read_buffer_desc.dsc$w_length = iosb.count;
            lib$put_output(&read_buffer_desc);
        }
    }
}
```

Example 4–3 assigns a write only channel to the mailbox.

Example 4–3 Mailbox Driver Program Example 3

```
/*
 * C program to demonstrate the new features of the Mailbox driver
 * This program is a Mailbox WRITER.  It assigns a WRITE_ONLY channel to the
 * mailbox.  It's partner program is a Mailbox READER.
 */

#include "sys$library:descrip"          /* Descriptor structure definitions  */
#include "sys$library:libdef"           /* LIB RTL symbol definitions        */
#include "sys$library:rmsdef"           /* RMS status code def'ns            */
#include "sys$library:stdio"            /* Standard C symbol definitions     */
#include "decc$library:[include]ssdef"  /* System Service status code def'ns */
#include "decc$library:[include]cmbdef" /* CREMBX definitions                */
#include "decc$library:[include]agndef" /* ASSIGN definitions                */
#include "decc$library:[include]iodef"  /* I/O definitions                   */

#define $ARRAY_DESCRIPTOR(name,size,array_name)                   \
    char array_name[ size ];                                      \
    struct dsc$descriptor_s name =                                \
       { size, DSC$K_DTYPE_T, DSC$K_CLASS_S, array_name }
```

(continued on next page)

Example 4–3 (Cont.) Mailbox Driver Program Example 3

```
main()
{
#define max_msg_len 256
$DESCRIPTOR(mailbox_name_desc,"MAILBOX_EXAMPLE");
$DESCRIPTOR(prompt_string_desc,"DATA TO SEND TO MAILBOX
    (<CTRL Z> to terminate) >>>");
$ARRAY_DESCRIPTOR(write_buffer_desc,max_msg_len,write_buffer);
int status, mailbox_channel, wait_efn;
int true=1, false = 0;
int MORE_ROOM_AST();
struct io_status_block {                              /*  I/O status block */
    short int condition;
    short int count;
    int dev;
    } iosb;

/*
 * Create a permanent mailbox with a WRITEONLY channel.  It's logical name
 *  will be entered into the LNM$PERMANENT_MAILBOX logical name table.
 */

    SYS$CREMBX(1,&mailbox_channel,0,0,0,0,&mailbox_name_desc,CMB$M_WRITEONLY);

/*
 * Mark it for deletion
 */
    SYS$DELMBX(mailbox_channel);

/*
 * Loop forever, first waiting til a READ channel is assigned to the mailbox
 * and then write data until there is no more data to write.
 */
    while (TRUE)
    {
        /*
         * Wait for a READER to assign a channel.  If a READER is already
         * assigned, this will return immediatly.
         */
        status = SYS$QIOW (
            0,
            mailbox_channel,
            IO$_SETMODE|IO$M_READERWAIT,
            &iosb,
            0,0,
            0,0,0,0,0,0);
        while (status)
        {
            write_buffer_desc.dsc$w_length = max_msg_len;
            status = lib$get_input(
                &write_buffer_desc,
                &prompt_string_desc,
                &write_buffer_desc.dsc$w_length);
            /*
             * If at end of file (user typed <CTRL Z>)
             * then stop here.
             */
            if (status == RMS$_EOF) SYS$EXIT(SS$_NORMAL);
```

(continued on next page)

Example 4–3 (Cont.) Mailbox Driver Program Example 3

```
            /* Keep trying to write the message, until it fits in the mailbox
             * Note that if the NORSWAIT function modifier had been eliminated
             * below, then the ROOM_NOTIFY and the retry loop could have been
             * removed.  ROOM_NOTIFY was used in this example simply to show
             * its' use. It would be more appropriately used when the program
             * has other things it can be working on, as opposed to the
             * example below in which the program is not doing anything except
             * WAITING for room in the mailbox.
             */
            status = SS$_NOREADER;                /* Force attempt to WRITE */
            while (status != SS$_NORMAL)
                {
                status = SYS$QIOW(
                    0,
                    mailbox_channel,
                    IO$_WRITEVBLK|IO$M_READERCHECK|IO$M_NORSWAIT,
                    &iosb,
                    0,0,
                    write_buffer_desc.dsc$a_pointer, write_buffer_desc.dsc$w_length,
                    0,0,0,0);
                if (iosb.condition == SS$_NOREADER) SYS$EXIT(SS$_NOREADER);
                if (status == SS$_MBFULL)
                {
                    LIB$GET_EF(&wait_efn);
                    SYS$CLREF(wait_efn);
                    SYS$QIOW (
                        0,
                        mailbox_channel,
                        IO$_SETMODE|IO$M_MB_ROOM_NOTIFY,
                        &iosb,
                        0,0,
                        MORE_ROOM_AST,wait_efn,0,0,0,0);
                        SYS$WAITFR(wait_efn);
                }
            }
        }
    }
}

MORE_ROOM_AST(efn_to_set)
int efn_to_set;
{
    SYS$SETEF(efn_to_set);
}
```

5
Terminal Driver

This chapter describes the use of the terminal driver (TTDRIVER) and the LAT port driver (LTDRIVER). The terminal driver supports the asynchronous, serial line multiplexers listed in Table 5–1. 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 5.4.4).

5.1 Supported Terminal Devices

In addition to the multiplexers listed in Table 5–1, the terminal driver supports serial line interfaces. 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 operating system for this interface are included in Table 5–1.

The remote command terminal, used by the DCL command SET HOST, also makes use of the features listed in Section 5.2.

Table 5–1 Supported Terminal Devices

Terminal Interface	No.of Lines	Outout		Split Speed	Bus	International Modem Control
		Silo	DMA			
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
MicroVAX 2000	4	No	No	Yes	None	No
MicroVAX 3100	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 bus	Full
DHB32	16	No	Yes	Yes	VAXBI bus	Full

[1]Depends on whether the DHV or DHU mode is selected when the board is installed

(continued on next page)

Table 5–1 (Cont.) Supported Terminal Devices

Terminal Interface	No.of Lines	Outout Silo	Outout DMA	Split Speed	Bus	International Modem Control
DSH32	8	Yes	No	Yes	MicroVAX 2000, MicroVAX 3100	No
DMF32	8	Yes	Yes[2]	Yes[2]	UNIBUS	Yes
DMZ32	24	Yes	Yes	Yes	UNIBUS	Full
DZ11	8/16	No	No	No	UNIBUS	No
DZ32	8	No	No	Limited	UNIBUS	No
LAT	[3]	No	Yes	[3]	N/A	[3]
VAX 8200 serial lines	4	No	No	No[4]	None	No
VAXstation 3100	4	No	No	Yes	None	No
VAXstation 4000	2	No	No	Yes	None	No
DEC 2000 Model 300	2	No	No	No	None	Full
DEC 2000 Model 300[7]	4, 8	Yes	No	No	EISA	Full
Alpha Server 2100	2	No	No	No	None	Full
Alpha Server[7]	4, 8	Yes	No	No	EISA	Full
DEC 3000 Model 300	3	No	Yes[5]	No	None	Full
DEC 3000 Model 400	4	No	Yes[6]	No	None	Full
DEC 3000 Model 500	4	No	Yes[6]	No	None	Full
DEC 4000 Model 600	2	No	No	No	None	Full

[2]Lines 0 and 1.

[3]Server dependent.

[4]The 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 can include a LAT terminal interface, but do not include other terminal interfaces.

[5]Communications only if not booted as alternate console.

[6]Communications only.

[7]Optional multifunctional serial/parallel PC4XD-AA adapter card. You can daisy-chain up to four boards in one system, resulting in 16, 32, or 64 ports.

5.2 Terminal Driver Features

The 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 Institute (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 logout
  - Preservation of process across hangups
- Miscellaneous
  - Terminal/mailbox interaction
  - Autobaud detection
  - Out-of-band control character handling

5.2.1 Input Processing

The 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.

5.2.1.1 Command Line Editing and Command Recall

The terminal driver input process defines a bounded set of line editing functions. You can access these functions with control keys on all keyboards, and with some special keys on certain keyboards as well. You can move the cursor in single-character increments (left arrow or Ctrl/D, right arrow or Ctrl/F) or in multicharacter increments, to the 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 or overstrike mode is the terminal's default characteristic[1] at the beginning of a read operation, but you can change it with the toggle insert/overstrike key (Ctrl/A). You can delete characters in word increments (Ctrl/J or line feed) and beginning-of-the-line increments (Ctrl/U).

When you use the terminal driver's editing functions, the following restrictions result:

- You cannot move the cursor to a previous line after a line wrap.
- You cannot insert a character if the insertion would force a line wrap or if a tab follows the current cursor position.
- You cannot delete a word at the beginning of a line after a line wrap.

[1] Digital suggests that new users specify overstrike mode.

- You cannot assign the line editing function 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, you edit or reenter the line 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 OpenVMS Versions 3.0 through 3.7.

5.2.1.2 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 B–1 lists the numeric values for all control characters.)

You enter some control characters at the terminal by simultaneously pressing the Ctrl key and a character key, such as Ctrl/*x*. Table 5–2 lists the terminal control characters. You can change control character echo strings (Ctrl/C, Ctrl/Y, Ctrl/O, and Ctrl/Z) on a systemwide basis (see the *OpenVMS System Management Utilities Reference Manual*). You enter special keys, such as Return, Line Feed, and Escape, 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 *OpenVMS DCL Dictionary* for information on line editing function keys and the SET TERMINAL command.

Table 5–2 Terminal Control Characters

Control Character	Meaning
Cancel (Ctrl/C—F6[1])	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 5.4.3.2). 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. Additional consequences of Ctrl/C are as follows: • 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$_CONTROLC, or SS$_CONTROLY if Ctrl/C is converted to Ctrl/Y.
Delete character (DELETE)	Removes the last character entered from the input stream. 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)	Deletes 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$M_CANCTRLO modifier, or the receipt of the next Ctrl/O. The terminal echoes OUTPUT OFF. The current write operation (if any) and write operations performed while Ctrl/O is in effect are completed with a status return of SS$_CONTROLO. 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.

[1]F6 on the LK201 is Interrupt/Cancel.

(continued on next page)

Table 5–2 (Cont.) Terminal Control Characters

Control Character	Meaning
Interrupt (Ctrl/Y)	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$_SETMODE function code. The command interpreter's Ctrl/Y AST handler always takes precedence over a user program's Ctrl/Y AST handler. Entering Ctrl/Y results in an AST to an enabled process to signify that the user entered Ctrl/Y from the terminal. The terminal performs a carriage-return/line-feed combination, echoes INTERRUPT, and performs another carriage-return /line-feed combination if the AST and echo are enabled. Ctrl/Y is ignored (and not echoed) if the process is not enabled for the AST. Additional consequences of Ctrl/Y are as follows: • The type-ahead buffer is flushed. • Ctrl/S and Ctrl/O are reset. • All queued and in-progress write operations and all in-progress read operations are successfully completed with a 0 transfer count. The status return is SS$_ 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) (Backspace)	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.
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.

(continued on next page)

Table 5–2 (Cont.) Terminal Control Characters

Control Character	Meaning
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.

5.2.1.3 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 one of the DECforms, FMS, or TDMS layered products, which support the entire data entry environment. Section 5.4.1.4 describes the supported primitives.

5.2.1.4 Escape and Control Sequences

Escape and control sequences provide additional terminal control not furnished by the control characters and special keys (see Section 5.2.1.2). 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$M_ESCAPE (see Table 5–5) is used to specify that terminal lines can generate valid escape sequences. Also, the read function modifier IO$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 5.5).

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 <O> <20-2F>...<40-7E>
```

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 control 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 where 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 5.2.1.2 describes control character functions during escape sequences.

Table B–2 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 5–6). Table B–2 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 the various terminals). Unless otherwise noted, the operation of escape sequences and modes is identical to the particular terminals that implement these features.

5.2.1.5 Type-Ahead Feature

Input (data received) from a 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 5–5 and Section 5.4.3).

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 5–7).

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$M_HOSTSYNC (see Table 5–5 and Section 5.4.3). If TT$M_HOSTSYNC is not set, Ctrl/G (bell) is returned to inform you that the type-ahead buffer is full. The buffer must then be emptied, at which time a status of SS$_DATAOVERUN is returned. If TT$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_ALTYPEAHD 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, and the user has posted another READ QIO.

The type-ahead buffer length is variable, with possible values in the range of 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_ALTYPEAHD, which allows the use of a larger type-ahead buffer specified by the system generation parameters TTY_ALTYPAHD and TTY_ALTALARM. (TTY_ALTYPAHD 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$M_PASTHRU and the read function IO$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.

5.2.1.6 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 5.4.1.2).

All characters are 7-bit ASCII characters unless data is input on an 8-bit terminal (see Section 5.4.1). The characteristic TT$M_EIGHTBIT determines whether a terminal uses the 7-bit or 8-bit character set; see Table 5–5. All input characters (except some special keys; see Section 5.2.1.2) 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.

5.2.1.7 Special Operating Modes

The terminal driver supports many special operating modes for terminal lines. (Tables 5–5 and 5–6 in Section 5.3 list these modes.) All special modes are enabled or disabled by the set mode and set characteristics functions (see Section 5.4.3).

5.2.2 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) ouput, 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.

5.2.2.1 Duplex Modes

The 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.

The following terms describe the state of a read request:

- A read request is *active* when the terminal driver removes that request from the head of the I/O queue.
- A read request is *started* when the terminal driver moves the first character into the read buffer.

In the terminal driver, write requests usually have priority. A write request can interrupt an active, but not started, read request.

The terminal driver does not start a read request until all outstanding writes are completed. This means that a read request could be removed from the head of the read queue while write requests are outstanding, but the first character is not moved into the read buffer until all outstanding writes are completed.

Once a read request is started, 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.

5.2.2.2 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$M_NOFORMAT modifier and thereby reduce overhead. IO$M_NOFORMAT overrides all formatting except fallback translation. Setting the PASTHRU mode (TT2$M_PASTHRU) is equivalent to writing with the noformat modifier.

Fallback conversions occur regardless of formatting mode.

5.2.2.3 SET HOST Facility and Output Buffering

The SET HOST facility emulates the terminal driver in the way it writes data to the terminal by stopping the display as soon as the abort character is entered. However, the SET HOST facility behaves differently from the terminal driver in that it buffers output data from the program that is executing. Occasionally, this causes a perception problem for the user when the program is aborted with a Ctrl/C, Ctrl/Y, or an out-of-band abort character. The user expects the program to terminate and the display to stop immediately.

CTDRIVER and RTPAD

When used between two systems, the SET HOST facility consists of two components: RTPAD on the local node and CTDRIVER on the remote node. Both components buffer output data to enhance performance when using wide area networks. CTDRIVER performs the initial buffering, queues the buffers for network transfer, and returns a successful write status. The user should note that the local terminal display reflects the output of the executing program after the data has been buffered and transferred over the network—not as the output buffers are filled on the remote node.

The delay between execution of an application and the display of its output can lead to several anomalies in the effects of Ctrl/C, Ctrl/Y, and out-of-band abort characters.

Output Line Not in Sequence Following an Abort Character

After you enter an abort character (Ctrl/C, Ctrl/Y or an out-of-band abort character) that causes the input or output to be aborted, it is possible to receive an additional line of output. This occurs when the application program calls $QIO (either directly or indirectly through RMS or language support routines) to output data to a buffer *at the same time* the abort character is entered.

When CTDRIVER receives the abort character (Ctrl/C, Ctrl/Y, or an out-of-band abort character) from the network, it flushes the current output buffers and aborts any pending read operations. However, if the application program calls $QIO with a write operation when the abort character is entered, the $QIO write data is still buffered and then displayed. The data may not be the next output in sequence from the user's point of view, since all the previous output buffers in CTDRIVER were flushed and the data in them was not displayed.

When using the terminal driver, the effect of an abort character on the display screen is different. The terminal driver does not buffer output from the application during program execution. If the application program has just called $QIO with a write operation when the abort character is entered, then the $QIO write data is displayed. Because all write operations are sequential and do not complete until the output is actually displayed, the additional line displayed is in sequence. There is no break in the data. Normally, the user will not notice that there is an additional line.

Extra Input Prompt Following an Abort Character

For connections between systems, the CTERM protocol allows CTDRIVER to synchronize with RTPAD before displaying any more data on the terminal.

Note

Prior to VAX VMS Version 5.2, a control character entered during program execution to abort input and output could cause the system to display more than one input prompt.

If the SET HOST facility is used between systems running VMS version 5.2 and an earlier version, the extra input prompt is still displayed.

Processing Abort Characters

The abort character AST is delivered after the message describing the aborted read operation has been received. Thus, the read status should be set very shortly after the abort character AST is delivered to the application. Note, however, these are still two asynchronous events, and the application must still synchronize with the completing read operation.

Note

Prior to VAX VMS Version 5.2, if an application had a read operation pending and had queued a Ctrl/C, Ctrl/Y, and out-of-band abort character AST, it was possible to queue multiple read operations unknowingly when the read operation was aborted.

Captive Command Procedures and Ctrl/Y

CTDRIVER and RTPAD emulate the terminal driver in that the current read operation and all pending write operations abort when Ctrl/Y is entered. However, the pending write operations also include all the buffered output that occurred and that would have been output before the Ctrl/Y was entered but due to the buffering was not.

The effect of the buffering can be confusing if a Ctrl/Y is entered when a captive command procedure is executing. During execution of captive command procedures, DCL has a Ctrl/Y pending. When this AST is delivered, DCL only reenables it; no other action is performed. In that case, if the program being executed only performs output, it appears that the program was aborted by the Ctrl/Y. Actually, the program completed execution before the Ctrl/Y was entered, and the Ctrl/Y merely discarded all the buffered output.

5.2.3 Dial-Up Support

The 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 operating system.

5.2.3.1 Modem Signal Control

Dial-up lines with the characteristic TT$M_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 dialup monitoring period for multiplexers that do not support modem signal transition interrupts (see Table 5–1).

If a line's carrier signal is lost, the driver waits 2 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 International Telegraph and Telephone Consultative Committee (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 2 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.

Note

Some systems, such as the VAXstation 3100, provide built-in serial lines using 6-pin modular jacks. These lines do not provide the minimum required modem signals. Although the hardware may allow a dial-out connection to be established, hangup cannot be detected and process deletion will not occur on these lines.

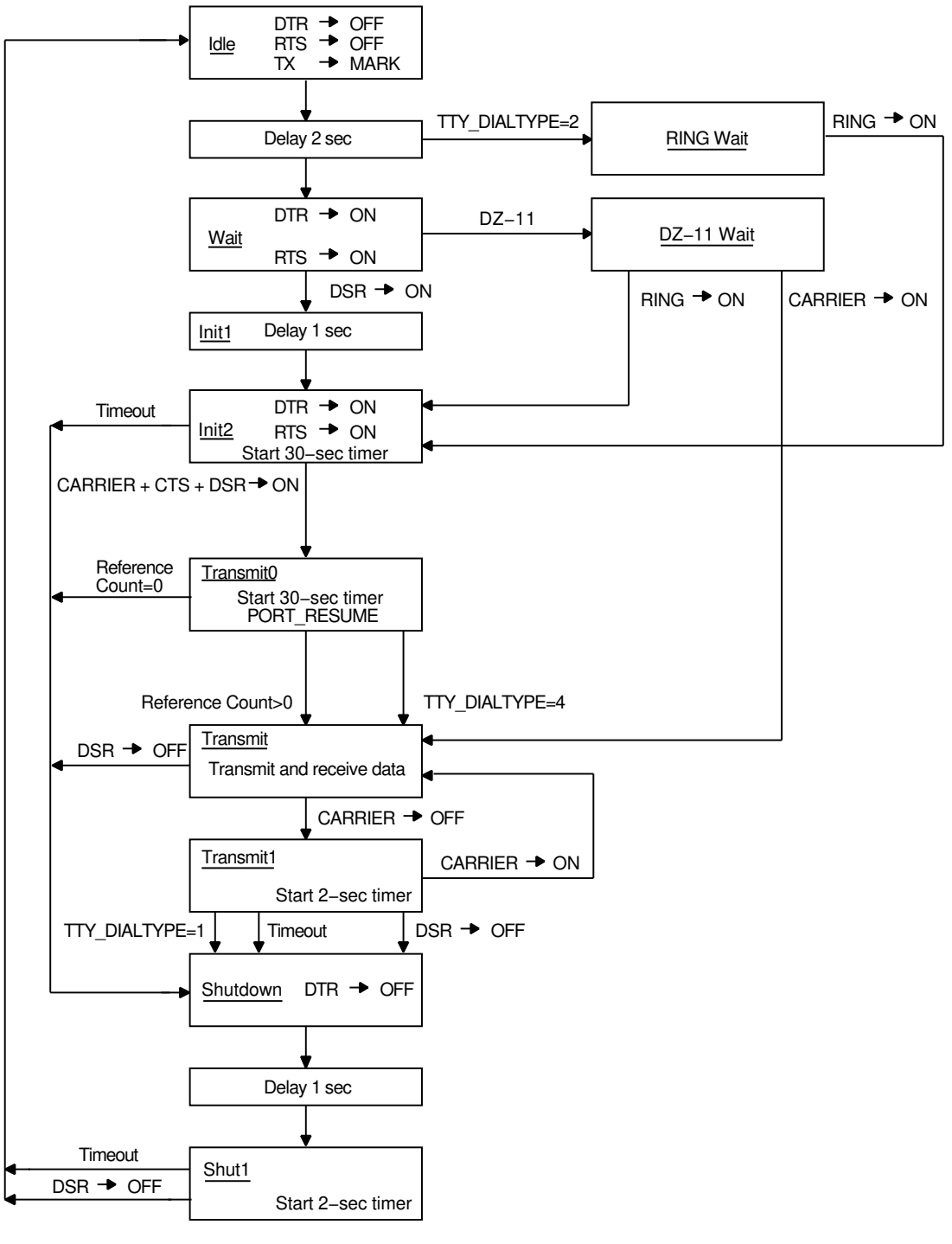
For terminals with the TT$M_MODEM characteristic, TT$M_REMOTE reflects the state of the carrier signal. TT$M_REMOTE is set when the carrier signal changes from off to on, and cleared when the carrier signal is lost.

A line that does not have TT$M_MODEM set does not respond to modem signals or set the DTR signal. Modem signals can be set and sensed manually through use of the IO$M_MAINT function modifier (see Section 5.4.3.3).

The 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 5–1).

Table 5–3 lists the control and data signals used in a full modem control mode configuration (in a two-way simultaneous, symmetrical transmit mode). Figure 5–1 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.

Figure 5–1 Modem Control: Two-Way Simultaneous Operation



ZK–0687–GE

Set mode function modifiers are provided to allow a process to activate or deactivate modem control signals (see Section 5.4.3.3).

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 dialup 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 dialup lines will skip the timeout waiting for a channel to be assigned.)

Table 5–3 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 operating system initiates a 30-second timer. This ensures that the phone line will be disconnected if CARRIER is not detected.

[1]Multiplexers (All = any supported controller; Full = DZ32, DMF32, DMB32, DMZ32, DHU11, DHV11, and CXY08)

(continued on next page)

Table 5–3 (Cont.) Control and Data Signals (Full Modem Mode Configuration)

Signal	Source	MUX[1]	Meaning
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.
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 operating system initiates a 30-second timer. This ensures that the phone line will be disconnected if CARRIER is not detected.

[1]Multiplexers (All = any supported controller; Full = DZ32, DMF32, DMB32, DMZ32, DHU11, DHV11, and CXY08)

5.2.3.2 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.

5.2.3.3 Preservation of a Process Across Hangups

Disconnectable terminals allow a connection to a physical terminal line to be broken without losing the job.

VAX

On VAX systems, the following SYSGEN command allows terminals to be disconnectable terminals:

```
SYSGEN>  CONNECT VTA0/NOADAPTER/DRIVER=TTDRIVER♦
```

Alpha

On Alpha systems, the following SYSMAN command allows terminals to be disconnectable terminals:

```
SYSMAN>  IO CONNECT VTA0/NOADAPTER/DRIVER=SYS$TTDRIVER♦
```

On VAX and Alpha systems, 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.
- 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.

5.2.4 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 4 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:

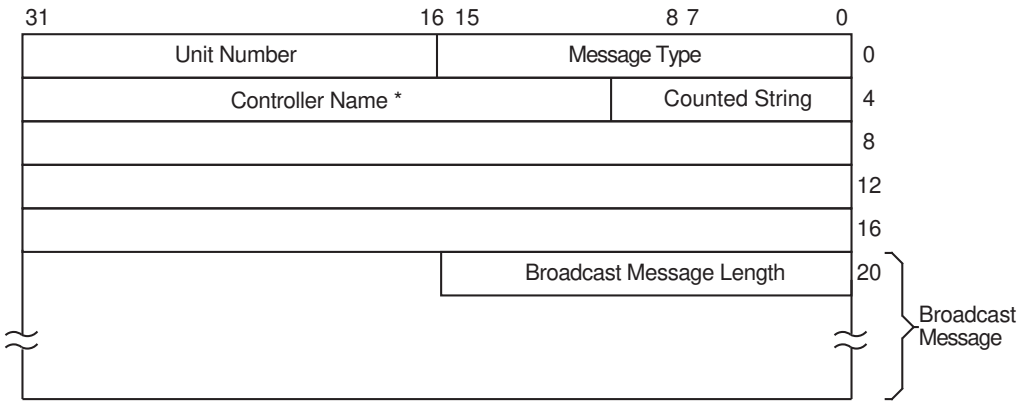
- Unsolicited data in the type-ahead buffer. The use of the associated mailbox can be enabled and disabled as a subfunction of the read and write requests (see Sections 5.4.1 and 5.4.2). (Initially, mailbox messages are enabled on all terminals. This is the default state.) Thus, the user process can 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 5–2):

- 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.
- Counted string to specify the device name.
- Controller name.
- Message (for broadcasts).

Figure 5–2 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 4 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.

5.2.5 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.

The autobaud function works with either even parity or no parity, but not with odd parity. If a line is set to even parity and has 7 bits of data, the line automatically switches to no parity if a terminal not generating parity attempts to log in.

The SET TERMINAL qualifier /EIGHT_BIT specifies that the terminal uses 8-bit ASCII code. /NOEIGHT_BIT, which is the default, specifies 7-bit ASCII code. (If parity is specified, the parity bit is separate from the data bits.) The optimal settings for automatic baud rate detection on Digital terminals are /NOEIGHT_BIT/PARITY=EVEN or /EIGHT_BIT/NOPARITY, although automatic baud rate detection also works with other combinations, such as /NOEIGHT_BIT /NOPARITY.

Table 5–6 describes the terminal characteristic TT2$M_AUTOBAUD, which allows the baud rate to be set automatically at login.

Digital does not usually recommend specifying the /FRAME qualifier with the SET TERMINAL command. The terminal driver selects the frame size (the number of data bits that the device can transmit) based on how the /PARITY and /EIGHT_BIT qualifiers are set. It might be necessary to change these values if the terminal is not made by Digital.

5.2.6 Out-of-Band Control Character Handling

All control characters (0 through 1F hexadecimal) can be enabled as out-of-band characters. Typing one of these characters immediately delivers an AST to the requesting process. DCL uses this mechanism to sense whether Ctrl/T has been entered. Out-of-band character options allow using the IO$M_INCLUDE function modifier to include the character in the data stream and the IO$M_TT_ABORT function modifier to abort the current input or output operation.

5.3 Terminal Driver Device Information

You can obtain information on terminal characteristics by using the Get Device /Volume Information ($GETDVI) system service. (See the *OpenVMS System Services Reference Manual*.) The sense mode function provides an alternative means to obtain terminal characteristics; see Section 5.4.5.

$GETDVI returns terminal characteristics when you specify the item codes DVI$_DEVCHAR, DVI$_DEVDEPEND, and DVI$_DEVDEPEND2. Tables 5–4, 5–5 and 5–6 list these characteristics. Terminal characteristics are normally set during system generation to any one of, or a combination of, the values listed in Table 5–5. DVI$_DEVDEPEND returns a longword field in which the three low-order bytes contain the device-dependent characteristics and the high-order byte contains the page length. Page length can have a value in the range of 0 through 255. The $DEVDEF macro defines the device-independent characteristics, the $TTDEF macro defines the device-dependent characteristics, and the $TT2DEF macro defines the extended device-dependent characteristics.

DVI$_DEVCLASS and DVI$_DEVTYPE return the device class and device type names, which are defined by the $DCDEF and $TTDEF macros, respectively. The device class for terminals is DC$_TERM. The terminal model determines the device type. For example, the device type for the VT240 is TT$_VT200_SERIES. DVI$_DEVBUFSIZ returns the page width, which can be a value in the range of 1 through 511. The driver does not accept a value of 0.

Table 5–4 Terminal Device-Independent Characteristics

Characteristic	Meaning
DEV$M_AVL	Terminal is on line and available.
DEV$M_CCL	Carriage control is enabled.
DEV$M_DET	Terminal is detached.
DEV$M_IDV	Terminal is capable of input.
DEV$M_ODV	Terminal is capable of output.
DEV$M_OPR	Terminal is enabled as an operator console.
DEV$M_REC	Device is record-oriented.
DEV$M_RTT	Terminal has remote terminal UCB extension.
DEV$M_SPL	Device is spooled.
DEV$M_TRM	Device is a terminal.
DEV$M_NET	Terminal line is allocated for DECnet use

Table 5–5 Terminal Characteristics

Value[1]	Meaning
TT$M_CRFILL	Terminal requires fill after the Return key is pressed (the fill type can be specified by the set mode function P4 argument).
TT$M_EIGHTBIT	Terminal uses the 8-bit ASCII character set (see Appendix B). Terminals without this characteristic use the 7-bit ASCII code. In this case, the eighth bit is masked out on received characters and is ignored on output characters. The eighth bit is meaningful only if TT$M_EIGHTBIT is set.
TT$M_ESCAPE	Terminal generates escape sequences (see Section 5.2.1.4). Escape sequences are validated for syntax.
TT$M_HALFDUP	Terminal is in half-duplex mode (see Section 5.2.2.1). All read and write requests are executed sequentially.
TT$M_HOSTSYNC	The host system is synchronized to the terminal. Ctrl/Q and Ctrl/S are used to control data flow and thus keep the type-ahead buffer from filling. TT$M_HOSTSYNC should always be set on LAT terminals.
TT$M_LFFILL	Terminal requires fill after the line-feed character is processed. (The fill can be specified by the set mode P4 argument.)
TT$M_LOWER	Terminal has the lowercase character set. Unless the terminal is in the PASTHRU mode or IO$M_NOFORMAT is specified, all input and echoed lowercase characters (hexadecimal 61 to 7A) are converted to uppercase if TT$M_LOWER is not set. (The character ALTMODE (decimal 125 and 126, or hexadecimal 7D and 7E) converts to ESCAPE on terminals that do not have the lowercase characteristic TT$M_LOWER set.)

[1]Defined by the $TTDEF macro. The prefix can be TT$M_ or TT$V_. TT$M_ is a bit mask whose bit corresponds to the specific field; TT$V_ is a bit number.

(continued on next page)

Table 5–5 (Cont.) Terminal Characteristics

Value[1]	Meaning
TT$M_MBXDSABL	Mailboxes associated with the terminal do not receive notification of unsolicited input or hangup (see Section 5.2.3). This bit can be set by the IO$M_DSABLMBX function modifier for read requests and cleared by the IO$M_ENABLMBX function modifier for write requests.
TT$M_MECHFORM	Terminal has mechanical form feed. The terminal driver passes form feeds directly to the terminal instead of expanding to line feeds.
TT$M_MECHTAB	Terminal has mechanical tabs and is capable of tab expansion. To accomplish correct line wrapping, the terminal driver assumes there are eight spaces between tab stops.
TT$M_MODEM	Terminal line is connected to a modem. If TT$M_MODEM is set, the terminal driver automatically handles modem control. If TT$M_MODEM is not set, all modem signals are ignored. If TT$M_MODEM is set and then cleared, a hangup is declared on the terminal line if that line is in the remote state (TT$M_REMOTE is set). If DTR and RTS are set with IO$_SETMODE!IO$M_SET_MODEM!IO$M_MAINT on a nonmodem port, DTR and RTS goes off and then back on when the port is set for modem. TT$M_MODEM is not supported for LAT devices.
TT$M_NOBRDCST	Terminal does not receive any broadcast messages.
TT$M_NOECHO	Input characters are not echoed on this terminal line (see Section 5.2.1.5).
TT$M_NOTYPEAHD	Data must be solicited by a read operation. Data is lost if received in the absence of an outstanding read request (if it is unsolicited data). Disables type-ahead feature (see Section 5.2.1.5). If this characteristic is set, login attempts on this line are disabled. See Section 5.2.3.1 for information on modem signal control.
TT$M_READSYNC	Read synchronization is enabled. The host explicitly solicits all read operations by entering a Ctrl/Q and terminates the operation by entering a Ctrl/S. TT$M_READSYNC is not applicable to LAT terminals.
TT$M_REMOTE	Dial-up characteristic is enabled. The terminal returns to local mode when a hangup occurs on the terminal line (see Section 5.2.3). This characteristic cannot be changed; it is only informational.
TT$M_SCOPE	Terminal is a video screen display (CRT terminal), for example, the VT100 or VT240 terminals.
TT$M_TTSYNC	The terminal is synchronized to the host system. Output to the terminal is controlled by terminal-generated Ctrl/Q and Ctrl/S. TT$M_TTSYNC is not applicable to LAT terminals unless TT$M_PASTHRU is set and TT$M_TTSYNC is disabled, in which case the LAT session is placed in PASSALL mode.

[1]Defined by the $TTDEF macro. The prefix can be TT$M_ or TT$V_. TT$M_ is a bit mask whose bit corresponds to the specific field; TT$V_ is a bit number.

(continued on next page)

Table 5–5 (Cont.) Terminal Characteristics

Value[1]	Meaning
TT$M_WRAP	A carriage-return/line-feed combination should be inserted if the cursor moves beyond the right margin. If TT$M_WRAP is not set, no carriage-return/line-feed combination is sent. The operating system does not support hardware-provided wrapping functions.

[1]Defined by the $TTDEF macro. The prefix can be TT$M_ or TT$V_. TT$M_ is a bit mask whose bit corresponds to the specific field; TT$V_ is a bit number.

Table 5–6 Extended Terminal Characteristics

Value[1]	Meaning
TT2$M_ALTYPEAHD	Alternate type-ahead buffer size is enabled. Use the alternate type-ahead buffer size specified during system generation (see Section 5.2.1.5). If a type-ahead buffer already exists for a terminal line, there is no effect when this characteristic is set for that line. TT2$M_ ALTYPEAHD should be set prior to using the terminal, such as in the startup command procedure. You can only set TT2$M_ALTYPEAHD; this characteristic cannot be cleared until the system is rebooted.
TT2$M_ANSICRT	ANSI CRT terminal is enabled. This characteristic is set by the SET TERMINAL command. TT2$M_ ANSICRT is a subset of the ANSI standard with no Digital-private escape sequences (see Appendix B). It is also a subset of the VT100-family terminals (because TT2$M_ANSICRT is a subset of TT2$M_DECCRT) and the VT100. Terminals with this characteristic must provide a display of at least 24 lines, each with 80 columns.
TT2$M_APP_KEYPAD	Notifies application programs of state to set the keypad to when exiting.
TT2$M_AUTOBAUD	Automatic baud rate detection is enabled. This characteristic allows the baud rate to be set automatically when you log in. (The baud rate is set when one or more carriage returns are entered during the login procedure.) Terminals are set to a permanent speed of 9600 baud. If TT2$M_AUTOBAUD is specified, the permanent speed must not be changed while this characteristic is in use on a given terminal line. See Section 5.2.5 for additional information on automatic baud rate detection.
TT2$M_AVO	Advanced video is enabled. This characteristic provides the terminal with blink, bold, and flashing fields as well as a full screen of 132 character lines. TT2$M_AVO is set by the SET TERMINAL command. Appendix B lists the valid escape sequences for terminals with the TT2$M_AVO characteristic.

[1]Defined by the $TT2DEF MACRO. The prefix can be TT2$M_ or TT2$V_. TT2$M_ is a bit mask in which the bit set corresponds to the specific field; TT2$V_ is a bit number.

(continued on next page)

Table 5–6 (Cont.) Extended Terminal Characteristics

Value[1]	Meaning
TT2$M_BLOCK	Block mode is enabled. This characteristic is set by the SET TERMINAL command. TT2$M_BLOCK defines additional ANSI-defined and Digital-private escape sequences (see Appendix B). Terminals with this characteristic are capable of local editing and block mode transmission (XON/XOFF flow control must be honored), and have protected fields. If the terminal is used for large amounts of block input, TT2$M_ ALTYPEAHD should also be specified.
TT2$M_BRDCSTMBX	Mailbox broadcasts messages. Broadcast messages are sent to an associated mailbox, if one exists.
TT2$M_COMMSYNC	Enables devices such as asynchronous printers to be connected to terminal ports. Flow control is handled by EIA modem signals, instead of XON/XOFF. Setting TT2$M_COMMSYNC activates the DTR and RTS signals; data is sent once the DSR and CTS signals are also present. If either of these signals is not present, printing stops. When both signals are present again, printing resumes. Do not set TT2$M_COMMSYNC on a line connected to a modem that is intended for interactive use. TT2$M_COMMSYNC disables the modem terminal characteristic that disconnects a user process from the terminal line in case of a modem phone line failure. With TT2$M_COMMSYNC set, the next call on the terminal line could be attached to the previous user's process. TT2$M_COMMSYNC should also not be used in combination with XON/XOFF, TT$M_TTSYNC, or TT$M_HOSTSYNC. TT2$M_COMMSYNC and TT$M_ MODEM are mutually exclusive.
TT2$M_DECCRT	Digital CRT terminal. This characteristic is set by the SET TERMINAL command for all terminals that are upward-compatible with VT100-family terminals. TT2$M_DECCRT is a superset of TT2$M_ANSICRT. Additional ANSI-defined as well as most Digital-private escape sequences are allowed for terminals with this characteristic (see Appendix B); maintenance modes, VT52 mode, and the use of the LED displays are not defined by TT2$M_DECCRT. Not all VT100-family terminals implement these features. The presence of the advanced video feature cannot be assumed because it is a VT100 option. This restricts the use of graphics attributes. However, the TT2$M_AVO characteristic can be used to determine whether additional graphic attributes are available.
TT2$M_DECCRT2	Digital CRT terminal. This characteristic is set by the SET TERMINAL command for all terminals that are upward-compatible with VT200-family terminals. TT2$M_DECCRT2 is a superset of TT2$M_DECCRT.

[1]Defined by the $TT2DEF MACRO. The prefix can be TT2$M_ or TT2$V_. TT2$M_ is a bit mask in which the bit set corresponds to the specific field; TT2$V_ is a bit number.

(continued on next page)

Table 5–6 (Cont.) Extended Terminal Characteristics

Value[1]	Meaning
TT2$M_DECCRT3	Digital CRT terminal. This characteristic is set by the SET TERMINAL command for all terminals that are upward-compatible with VT300-family terminals. TT2$M_DECCRT3 is a superset of TT2$M_DECCRT2.
TT2$M_DECCRT4	Digital CRT terminal. This characteristic is set by the SET TERMINAL command for all terminals that are upward-compatible with VT400-family terminals. TT2$M_DECCRT4 is a superset of TT2$M_DECCRT3.
TT2$M_DIALUP	Terminal is a dialup line. Used by LOGINOUT for the disable dialup control.
TT2$M_DISCONNECT	Allows terminal disconnect when a hangup occurs (that is, when modem signals are lost, when the DCL commands DISCONNECT, or CONNECT /CONTINUE are entered, or when the BREAK key is pressed on a terminal that has the TT2$M_SECURE characteristic). These terminals are created as VTAn:. (See the description for the DCL command CONNECT /DISCONNECT in the *OpenVMS DCL Dictionary*.)
TT2$M_DMA	Direct memory access (DMA) mode. This characteristic enables the use of DMA mode for asynchronous DMA multiplexers. It is ignored by non-DMA controllers.
TT2$M_DRCS	Terminal supports loadable character fonts. This characteristic is set with the DCL command SET TERMINAL/SOFT_CHARACTERS.
TT2$M_EDIT	Terminal edit. This characteristic is set by the SET TERMINAL command for all terminals that support ANSI-defined advanced editing functions. These functions include the ability to insert or delete a line and the ability to insert or delete characters in an existing line. Terminals with this characteristic are a superset of TT2$M_DECCRT. Appendix B lists the valid escape sequences for terminals with the TT2$M_EDIT characteristic.
TT2$M_EDITING	Line editing is allowed.
TT2$M_FALLBACK[2]	Output is transformed from the 8-bit multinational character set to a 7-bit ASCII character set on terminals that do not support the 8-bit character set (see Appendix B).
TT2$M_HANGUP	Terminal hangup. Terminal lines connected through modems are hung up when a process logs out or is deleted. The state of this characteristic cannot be changed unless TT2$M_MODHANGUP is enabled or the process has either LOG_IO or PHY_IO privilege.
TT2$M_INSERT	Sets default mode for insert or overstrike at the beginning of each read operation.

[1]Defined by the $TT2DEF MACRO. The prefix can be TT2$M_ or TT2$V_. TT2$M_ is a bit mask in which the bit set corresponds to the specific field; TT2$V_ is a bit number.

[2]If an attempt is made to turn on TT2$V_FALLBACK for a disconnected virtual terminal (_VTAx:) or if the Terminal Fallback Facility (TFF) has not been activated, the status code SS$_BADPARAM is returned. For more information on TFF, refer to the *OpenVMS Terminal Fallback Utility Manual*.

(continued on next page)

Table 5–6 (Cont.) Extended Terminal Characteristics

Value[1]	Meaning
TT2$M_LOCALECHO	Local echo. This characteristic is used with TT$M_NOECHO. If both characteristics are set, only terminators and special control characters are echoed. Use of this mode is restricted to command line read operations. Application programs that use the IO$M_NOECHO function modifier will not necessarily work if TT2$M_LOCALECHO is set. Local echo is also not compatible with line editing (TT2$M_EDITING).
TT2$M_MODHANGUP	Modify hangup. If specified, TT2$M_HANGUP can be modified without privilege. Otherwise, logical or physical I/O privilege is required.
TT2$M_PASTHRU	Terminal is in PASTHRU mode; all input and output data is in 7- or 8-bit binary format (no data interpretation occurs). Data is terminated when the buffer is full or when the data that is read matches the specified terminator. If the characteristic TT$M_TTSYNC is set, Ctrl/S and Ctrl/Q interpretation does occur.
TT2$M_PRINTER	Digital CRT terminal with a local printer port.
TT2$M_REGIS	ReGIS graphics. The terminal supports the ReGIS graphics instruction set.
TT2$M_SIXEL	SIXEL graphics. The terminal supports the SIXEL graphics instruction set.
TT2$M_SECURE	For use with nonmodem, nonautobaud lines. This characteristic guarantees that no process is connected to the terminal after the BREAK key is pressed. If TT2$M_SECURE is not set, BREAK is a null key.
TT2$M_SETSPEED	Set speed. If specified, either LOG_IO or PHY_IO privilege is required to change terminal speed. TT2$M_SETSPEED is not supported for LAT devices.
TT2$M_SYSPWD	System password. This characteristic specifies that the login procedure should require the system password before the user name prompt is displayed.
TT2$M_XON	XON/XOFF control. If a set mode function is performed on a terminal in the Ctrl/S state, and if TT2$M_XON is set, output is resumed. Users should note that the driver will attempt to resume stopped (XOFF) output on the line. However, restarting the output may not be successful in all cases. The XON/XOFF feature does not work on all terminals, for example, the VT220.

[1]Defined by the $TT2DEF MACRO. The prefix can be TT2$M_ or TT2$V_. TT2$M_ is a bit mask in which the bit set corresponds to the specific field; TT2$V_ is a bit number.

5.3.1 Terminal Characteristics Categories

The set mode and set characteristics functions (see Section 5.4.3) and the DCL command SET TERMINAL are used to change terminal characteristics. The *OpenVMS DCL Dictionary* describes the SET TERMINAL command.

To customize terminal behavior and usage, the operating system divides terminal characteristics into the following categories:

- Format effectors—The following characteristics allow the user to specify terminal-dependent formatting requirements:

TT$M_CRFILL	TT$M_EIGHTBIT	TT$M_LFFILL
TT$M_LOWER	TT2$M_LOCALECHO	TT$M_MECHFORM
TT$M_MECHTAB	TT$M_NOECHO	TT$M_SCOPE
TT$M_WRAP		

- Generic terminal capabilities—The following characteristics specify generic terminal features available to applications programs:

TT2$M_ANSICRT	TT2$M_AVO	TT2$M_BLOCK
TT2$M_DECCRT	TT2$M_DECCRT2	TT2$M_DECCRT3
TT2$M_DECCRT4	TT2$M_DRCS	TT2$M_EDIT
TT2$M_PRINTER	TT2$M_REGIS	TT2$M_SIXEL

  Their use allows execution of these programs without knowledge of the actual terminal type. For example, a program should check for TT2$M_DECCRT rather than for VT100 or VT101.

- Protocol—The following characteristics control protocols used by the terminal:

TT$M_ESCAPE	TT$M_HALFDUP	TT$M_HOSTSYNC
TT2$M_PASTHRU	TT$M_TTSYNC	

- System management—The following characteristics, normally set only at system startup, allow the system manager to regulate terminal usage:

TT2$M_ALTYPEAHD	TT2$M_AUTOBAUD	TT2$M_DIALUP
TT2$M_DISCONNECT	TT2$M_DMA	TT2$M_HANGUP
TT$M_MODEM	TT$M_NOTYPEAHD	TT2$M_MODHANGUP
TT2$M_SECURE	TT2$M_SETSPEED	TT2$M_SYSPWD
TT2$M_COMMSYNC		

- User preference—The following characteristics allow you to customize the terminal operating mode:

TT2$M_APP_KEYPAD	TT2$M_FALLBACK	TT2$M_EDITING
TT2$M_INSERT	TT$M_NOBRDCST	

- Miscellaneous—The following characteristics provide greater program control of terminal operations:

TT2$M_BRDCSTMBX	TT$M_MBXDSABL	TT2$M_XON

5.4 Terminal Function Codes

The basic terminal I/O functions are read, write, set mode, set characteristics, sense mode, and sense characteristics. All I/O functions can take function modifiers.

5.4.1 Read

When a read function code is issued, the user-specified buffer is filled with characters from the associated terminal. The operating system provides the following read function codes:

- IO$_READVBLK—Read virtual block
- IO$_READLBLK—Read logical block
- IO$_READPROMPT—Read with prompt

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

- The user buffer is full.
- The received character is included in a specified terminator mask (see Section 5.4.1.2).

The following device- or function-dependent arguments are used with the read function codes. The codes can take all six arguments (P1 through P6) on QIO requests. The descriptions for these arguments differ when itemlist read operations are performed (see Section 5.4.1.3).

- P1—The starting virtual address of the buffer that is to receive the data read.
- P2—The size of the buffer that is to receive the data read in bytes. (A system generation parameter, MAXBUF, limits the maximum size of the buffer.)
- P3—Read with timeout, timeout count (see Table 5–7, IO$M_TIMED).
- P4—The read terminator descriptor block address (see Section 5.4.1.2).
- P5—The starting virtual address of the prompt buffer that is to be written to the terminal; for read with prompt operations using the IO$_READPROMPT function code. (This argument is specified as a value, rather than an address as in the P1 argument.)
- P6—The size of the prompt buffer that is to be written to the terminal; for read with prompt operations using the IO$_READPROMPT function code.

In a read with prompt operation, the P5 and P6 arguments specify the address and size of a prompt string buffer containing data to be written to the terminal before the input data is read. In a read with prompt operation, both read and write operations are performed on the specified terminal. The prompt string buffer is formatted like any other write buffer. If cursor position specifiers are supplied, they are not interpreted by the driver but passed to the terminal.

During a read with prompt operation, pressing Ctrl/O (which is turned off at the start of any read operation) stops the prompt string. If you press either Ctrl/U or Ctrl/X, the entire prompt string is written out again, and the current input is ignored. If you press Ctrl/R, the current prompt string and input are written to the terminal.

Depending on the terminal type and your input, the prompt string can be very simple or quite complex—from single command prompts to screen fills followed by input data. Digital recommends that prompt strings contain only one leading line feed.

In PASTHRU mode, data received from the associated terminal is placed in the user buffer as binary information without interpretation. (Prompts are not refreshed after a broadcast in PASTHRU mode.)

5.4.1.1 Function Modifier Codes for Read QIO Functions

Eight function modifiers can be specified with IO$_READVBLK, IO$_READLBLK, and IO$_READPROMPT. Table 5–7 lists these function modifiers and IO$_EXTEND, which is described in section 5.4.1.3. All read function modifiers are supported for LAT devices.

Table 5–7 Read QIO Function Modifiers for the Terminal Driver

Code	Consequence
IO$M_CVTLOW	Lowercase alphabetic characters (hexadecimal 61 to 7A) are converted to uppercase when transferred to the user buffer or echoed. This characteristic is used only for IO$_READLBLK, IO$_READVBLK, and IO$_READPROMPT.
IO$M_DSABLMBX	The mailbox is disabled for unsolicited data.
IO$M_ESCAPE	A valid ANSI escape sequence is recognized as a valid delimiter for the read operation. The TT$M_ESCAPE characteristic is overridden by this modifier for the current read operation.
IO$M_EXTEND	This characteristic provides additional functionality for read operations (see Section 5.4.1.3). Do not specify IO$M_EXTEND with other function modifiers.
IO$M_NOECHO	Characters are not echoed as they are entered at the keyboard. The terminal line can also be set to a ''no echo'' mode by the set mode characteristic TT$M_NOECHO, which inhibits all read operation echoing. Setting IO$M_NOECHO also disables line editing.
IO$M_NOFILTR	The terminal does not interpret Ctrl/U, Ctrl/R, or DEL. They are passed to the user. IO$M_NOFILTR explicitly disables line editing.
IO$M_PURGE	The type-ahead buffer is purged before the read operation begins.
IO$M_TIMED	The P3 argument specifies the maximum time (seconds) that can elapse between characters received from the terminal (the timeout value for the operation). Because driver timing operates on a 1-second timer, a 2-second timeout must be specified to guarantee a 1-second wait. The timer starts when the prompt echo is started. If the read time exceeds the time specified in P3, a timeout error (SS$_TIMEOUT) is returned in the read IOSB. All input characters received before the read operation timed out are returned in the user's buffer. A read with timeout operation, in which the timeout value is 0, empties the type-ahead buffer into the user buffer until the proper termination condition is reached (buffer full, termination character detected, or type-ahead buffer empty). The read operation then returns the count of characters read and, if a terminator is read, SS$_NORMAL; SS$_TIMEOUT is returned if no terminator is read. In either case the offset to terminator in the IOSB always indicates the number of characters read. If a write request is active and there is no prompt string, the read request generally times out with zero bytes of data being returned. If a read operation is interrupted by either a broadcast write or a synchronous write request, the timer operation is restarted.

(continued on next page)

Table 5–7 (Cont.) Read QIO Function Modifiers for the Terminal Driver

Code	Consequence
IO$M_TRMNOECHO	The termination character (if any) is not echoed. There is no formal terminator if the buffer is filled before the terminator is typed.

5.4.1.2 Read Function Terminators

The P4 argument to a read QIO function either specifies the terminator set for the read function or points to the location containing the terminator set. If P4 is 0, all ASCII characters with a code in the range 0 through 31 (hexadecimal 0 through 1F) except LF, VT, FF, TAB, and BS, are terminators (see Appendix B). This is the RMS standard terminator set. The delete character (hexidecimal 7F) and 8-bit controls in the range 128 through 159, and 255 (hexidecimal 80 through 9F, and FF) are also terminators. If line editing is enabled, only Return, Ctrl/Z, or an escape sequence terminates a read operation.

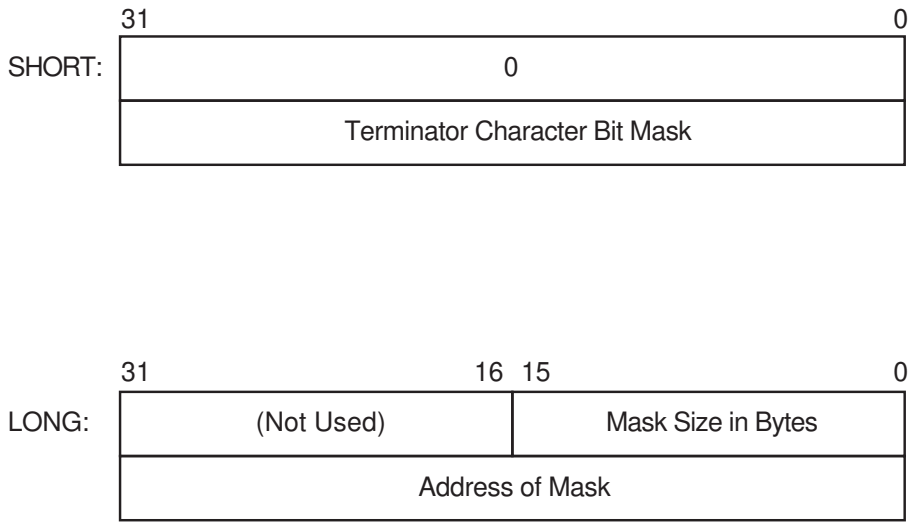
If P4 does not equal 0, it contains the address of a quadword that either specifies a terminator character bit mask or points to a location containing that mask. (Note that if P4 references an address in a MACRO program, a number sign (#) must precede the address, for example, P4=#TMASK.) The quadword has a short form and a long form, as shown in Figure 5–3. In the short form, the correspondence is between the bit number and the binary value of the character; the character is a terminator if the bit is set. For example, if bit 0 is set, NULL is a terminator; if bit 9 is set, TAB is a terminator. If a character is not specified, it is not a terminator. Since ASCII control characters are in the range 0 through 31, the short form can be used in most cases.

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

If the terminator mask is all zeros, there are no specified terminators. The read operation ends when the specified number of bytes (characters) have been transferred to the input buffer.

Certain control keys will not act as terminators unless IO$M_NOFILTR is specified or the line has the TT2$M_PASTHRU characteristic (see Section 5.2.1.2.).

Figure 5–3 Short and Long Forms of Terminator Mask Quadwords



ZK–0689–GE

5.4.1.3 Itemlist Read Operations

Itemlist read operations provide expanded software features to read QIO requests. The operating system provides the following combination of function code and modifier:

IO$_READVBLK!IO$M_EXTEND—Itemlist read virtual block

No other function modifiers can be specified in an itemlist read request.

Note

Itemlist read features supported by the terminal driver are not supported by all DECnet terminal emulators.

The itemlist read function code and modifier combination takes the following device- or function-dependent arguments:

- P1—The starting virtual address of the buffer that is to receive the data read
- P2—The size of the buffer that is to receive the data read in bytes. If required, the P2 size includes additional space for an overflow buffer to hold an escape sequence terminator (see item code TRM$_ESCTRMOVR in Table 5–8).
- P3—The access mode at which the itemlist is to be probed (optional)
- P5—The address of the itemlist buffer
- P6—The length in bytes of the itemlist buffer

P4 is not meaningful for itemlist read operations. P5 points to a series of item descriptors. Figure 5–4 shows the format for these descriptors. You cannot repeat the same item code in the same item list.

Figure 5–4 Itemlist Read Descriptor

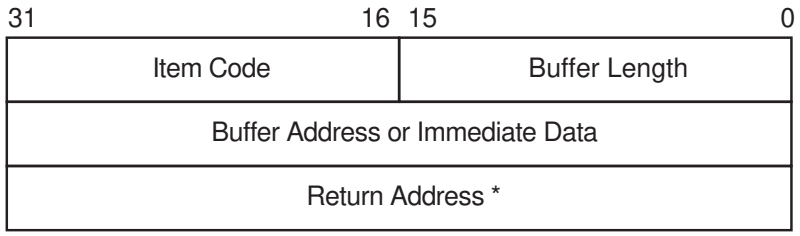


Table 5–8 lists the item codes that can be specified in the first longword of the item descriptors.

Table 5–8 Item Codes for Itemlist Read Operations for the Terminal Driver

Item Code	Meaning
TRM$_ALTECHSTR	Alternate echo string. The buffer length word contains the length of the string. The data address word contains the address of the string. The alternate echo string is written to the terminal after the first character is entered. This item code for character validating read mode (TRM$K_EM_RDVERIFY) editing only.
TRM$_EDITMODE	Extended editing modes. The immediate data longword specifies extended editing mode values. The buffer length word must be zero. The following editing modes are supported: TRM$K_EM_DEFAULT — Normal read mode. This is the default if TRM$_EDITMODE is not present in the itemlist. TRM$K_EM_RDVERIFY — Character validating read mode. See Section 5.4.1.4.
TRM$_ESCTRMOVR	Escape terminator overflow size. Specifies the number of bytes that may be used to hold an escape sequence terminator. This number should be included in P2, the buffer size argument, in addition to the space required for the data to be read. Note that this overflow area is for the terminator only; it is not available for user data. TRM$_ESCTRMOVR is useful in preventing partial escape errors, which return SS$_PARTESCAPE. This overflow buffer ensures that all the characters in an escape sequence terminator will fit in the user buffer, thus eliminating the need for additional single-character read operations.

(continued on next page)

Table 5–8 (Cont.) Item Codes for Itemlist Read Operations for the Terminal Driver

Item Code	Meaning
TRM$_FILLCHR	A 2-byte value that indicates the fill and clear character for TRM$K_EM_RDVERIFY. The first byte of the immediate data longword specifies the clear character; the second byte specifies the fill character. This item code is for character validating read mode (TRM$K_EM_RDVERIFY) editing only.
TRM$_INIOFFSET	Indicates the character in the initial string where echoing starts. The immediate data longword specifies the character.
TRM$_INISTRNG	Specifies a string to preload into the read buffer (P1). The buffer length word contains the length of the string. The data longword contains the address of the string. TRM$_INISTRNG must be specified if the edit mode is TRM$K_EM_RDVERIFY, and must be the same length as specified by TRM$_PICSTRNG.
TRM$_MODIFIERS	Read modifiers. The immediate data longword contains a 32-bit value that specifies modifiers to read operations. The read operations are defined in $TRMDEF. The buffer length word must be zero. The following bits are defined: TRM$M_TM_ARROWS — The terminal interprets the left and right arrow keys (TRM$K_EM_RDVERIFY mode only). The arrow keys are not put in the buffer and do not terminate the read. TRM$_ESCTRMOVR must be greater than or equal to 5. TRM$M_TM_AUTO_TAB — This bit creates an auto-tab mode field (TRM$K_EM_RDVERIFY mode only). TRM$M_TM_CVTLOW — Lowercase alphabetic characters (hexadecimal 61 to 7A) are converted to uppercase when transferred to the user buffer or echoed. TRM$M_TM_DSABLMBX — The mailbox is disabled for unsolicited data and for receiving hangup messages. TRM$M_TM_ESCAPE — A valid ANSI escape sequence is recognized as a valid delimiter for the read operation.

(continued on next page)

Table 5–8 (Cont.) Item Codes for Itemlist Read Operations for the Terminal Driver

Item Code	Meaning	
	TRM$M_TM_NOCLEAR	Fill characters are not replaced with clear characters after a nonfill character occurs (TRM$K_EM_RDVERIFY mode only).
	TRM$M_TM_NOECHO	Characters are not displayed as they are entered at the keyboard.
	TRM$M_TM_NOEDIT	This bit inhibits advanced editing for this read operation.
	TRM$M_TM_NOFILTR	The terminal does not interpret DEL, Ctrl/U, or Ctrl/R, but passes them to you. This characteristic explicitly disables line editing.
	TRM$M_TM_NORECALL	This bit inhibits command recall (Ctrl/B) by the terminal driver.
	TRM$M_TM_OTHERWAY	This bit sets left-justify fields to insert mode and right-justify fields to overstrike mode (TRM$K_EM_RDVERIFY mode only). TRM$M_TM_TOGGLE must equal 1.
	TRM$M_TM_PURGE	The type-ahead buffer is purged before the read operation begins.
	TRM$M_TM_R_JUST	This bit creates a right-justified field (TRM$K_EM_RDVERIFY mode only).
	TRM$M_TM_TERM_ ARROW	The read operation is terminated when the left arrow key is pressed at the left margin or when the right arrow key is pressed at the right margin (TRM$K_EM_RDVERIFY mode only). TRM$M_TM_ARROWS must be enabled.
	TRM$M_TM_TERM_DEL	The read operation is terminated when the DELETE key is pressed at the left margin (TRM$K_EM_RDVERIFY mode only).

(continued on next page)

Table 5–8 (Cont.) Item Codes for Itemlist Read Operations for the Terminal Driver

Item Code	Meaning	
	TRM$M_TM_TOGGLE	Enables Ctrl/A to function as a toggle key between insert mode and overstrike mode (TRM$K_EM_RDVERIFY mode only). Left-justify insert mode shifts characters to the right; right-justify insert mode shifts characters to the left. Shifted characters are not checked for validity in their new positions.
	TRM$M_TM_TIMED	TRM$_TIMEOUT specifies the maximum time (seconds) that can elapse between characters received from the terminal; that is, the timeout value for the operation. TRM$M_TM_TIMED is assumed set if TRM$_TIMEOUT is included in the itemlist. See the description of IO$M_TIMED in Table 5–7.
	TRM$M_TM_TRMNOECHO	The termination character (if any) is not displayed. There is no formal terminator if the buffer is filled before the terminator is typed.
		All other bits must be zero.
TRM$_PICSTRNG	Character validation string. The buffer length word contains the length of the string, which must be the same as the length specified by TRM$_INISTRNG. The data address word contains the address of the string. TRM$_PICSTRNG must be specified if the edit mode is TRM$K_EM_RDVERIFY. Note that this item code is for character validating read mode (TRM$K_EM_RDVERIFY) editing only. The format of the character validation string is 1 byte per input character. Each byte is a bit mask. The following values are provided:	

Value	Meaning
TRM$M_CV_UPPER	Uppercase alphabetic
TRM$M_CV_LOWER	Lowercase alphabetic
TRM$M_CV_NUMERIC	Numeric (0—9)
TRM$M_CV_NUMPUNC	Numeric punctuation (+ - .)
TRM$M_CV_PRINTABLE	Printable ASCII character
TRM$M_CV_ANY	Any character

If no values are set, the corresponding character specified by TRM$_INISTRNG is used. Appendix B lists the multinational character set.

(continued on next page)

Table 5–8 (Cont.) Item Codes for Itemlist Read Operations for the Terminal Driver

Item Code	Meaning
TRM$_PROMPT	Specifies a prompt string. The buffer length word contains the length of the prompt. The data address word contains the address of the prompt string. See Section 5.4.1 for information on how carriage control specifiers in a prompt string are handled.
TRM$_TERM	The buffer length word determines the format of the nondefault terminator mask. If the buffer length word is zero, then the data longword is used as a short form mask. If the buffer length word is nonzero, then a mask *n* bytes long is available at the specified address.
TRM$_TIMEOUT	Read timeout. See the description of IO$M_TIMED in Table 5–7.

5.4.1.4 Read Verify Function

When using the read verify function, the terminal driver performs input validation based on character attributes. (Read verification bypasses the optionally specified termination mask (TRM$_TERM).) Validation is performed one character at a time as data is entered. Invalid characters are not echoed, and cause the read operation to complete. It is then up to the application program to handle the error appropriately.

The initial string describes the initial contents of the input field. This string may consist of data and marker characters. The clear character is displayed on the screen for each occurrence of the fill character in the initial string buffer.

The picture string is a string of bytes where each byte corresponds to one character of the field being entered. Each byte specifies a mask of legal character types for that character position. If the byte is left as zero, then that position is a marker character, and the character from the initial string is echoed for that position.

For left-justified fields, the prompt data is output to the terminal, followed by an optional number (TRM$_INIOFFSET) of initial string characters. Leading marker characters are always output following the prompt, leaving the cursor at the leftmost data position. As each character is entered, it is validated and then echoed, advancing the cursor position. Additional marker characters are skipped as they are encountered. If an input character fails the validation, the read operation is completed with the invalid character as the terminator.

For right-justified fields, the prompt is output and is followed by the initial string. (In general, TRM$_INIOFFSET is set to the length of TRM$_INISTRNG for right-justified fields.) The cursor position remains one position to the right of the initial string. For proper operation, right-justified fields cannot have mixed picture definitions. After each character is input, the entire prompt and input fields are output. Therefore, the prompt should include a cursor positioning escape sequence.

The definition of full field is different for left- and right-justified read operations. For left-justified fields, full field is detected when the character corresponding to the last nonmarker position in the picture string has been entered. For right-justified fields, full field is detected when a character other than the fill character is shifted into the leftmost, nonmarker position in the field.

If the modifier TRM$M_TM_AUTO_TAB is set in TRM$_MODIFIERS, then detection of a full field terminates the read operation. In the event of autotab termination, the terminator character in the IOSB is null. If the autotab option is not selected, then termination occurs when one more character is typed to a full field. Applications can detect this condition when the terminating character index is one character beyond the end of the field. The extra character is reported as the terminator. In a left-justified field, the IOSB index to the terminator is zero-based; in a right-justified field, this index is one-based.

If a read verify function is interrupted by an asychronous write operation, the read verify is completed with status SS$_OPINCOMPL.

No line editing functions other than the delete character function are supported for read verify.

5.4.2 Write

Write operations display the contents of a user-specified buffer on the associated terminal. The operating system provides the following write I/O functions, which are listed with their function codes:

- IO$_WRITEVBLK—Write virtual block
- IO$_WRITELBLK—Write logical block
- IO$_WRITEPBLK—Write physical block

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 to the terminal.
- P2—The number of bytes that are to be written to the terminal. (A system generation parameter, MAXBUF, limits the maximum size of the buffer.)
- P4—Carriage control specifier except for write physical block operations. (Write function carriage control is described in Section 5.4.2.2.)

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

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

Unless TT$M_MECHFORM is specified, multiple line feeds are generated for form feeds. The number of line feeds generated depends on the current page position and the length of the page. By producing a carriage return after the last line feed, a form feed also moves the cursor to the left margin. Multiple spaces are generated for tabs if the characteristics of the selected terminal do not include TT$M_MECHTAB (this does not apply to write physical block operations). Tab stops occur every eight characters or positions.

CTDRIVER and Buffered Output

CTDRIVER, a component of the SET HOST facility, buffers output from remote terminals in order to package multiple output requests into a single network transfer. As a result, control is returned early to the user with a status of SS$_NORMAL when the output buffer has been filled and successfully queued.

Note that this output might not be displayed if the user enters an abort character or a Ctrl/O.

5.4.2.1 Function Modifier Codes for Write QIO Functions

Five function modifiers can be specified with IO$_WRITEVBLK, IO$_WRITELBLK, and IO$_WRITEPBLK. Table 5–9 lists these function modifiers. All write function modifiers are supported for LAT devices.

Table 5–9 Write QIO Function Modifiers for the Terminal Driver

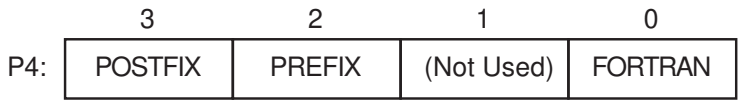
Code	Consequence
IO$M_BREAKTHRU	Allows breakthrough read regardless of the current active state.
IO$M_CANCTRLO	Turns off Ctrl/O (if it is in effect) before the write operation. Otherwise, the data cannot be displayed.
IO$M_ENABLMBX	Enables use of the mailbox associated with the terminal for notification that unsolicited data is available.
IO$M_NOFORMAT	Allows you to specify write functions without interpretation or format; in effect, the terminal line is in a temporary PASTHRU mode.
IO$M_REFRESH	If a read operation is interrupted by a write operation (by either a write breakthrough[1] or any other type of write), the terminal displays the current read data when the read function is restarted.

[1]Any interruption caused by the execution of the $BRDCST or the $BRKTHRU system service broadcasting messages to terminals is referred to as a ''write breakthrough.''

5.4.2.2 Write Function Carriage Control

The P4 argument is a longword that specifies carriage control. Carriage control determines the next printing position on the terminal. P4 is ignored in a write physical block operation. Figure 5–5 shows the P4 longword format.

Figure 5–5 P4 Carriage Control Specifier



ZK–0690–GE

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 5–10 lists the possible byte 0 values (in hexadecimal) and their meanings.

Table 5–10 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, print buffer contents, return[1])
30	0	Double-space carriage control. (Sequence: carriage-return/line-feed combination, carriage-return/line-feed combination, print buffer contents, return[1])
31	1	Page eject carriage control. (Sequence: form feed, print buffer contents, return)
2B	+	Overprint carriage control; allows double printing for emphasis or 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

[1]A carriage-return/line-feed combination is a carriage return followed by a line feed.

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 5–11 shows this encoding scheme in hexadecimal.

With several exceptions, Figure 5–6 shows the prefix and postfix hexadecimal coding that produces the carriage control functions listed in Table 5–10. Prefix and postfix coding provides an alternative way to achieve these controls.

In the first example in Figure 5–6, the prefix/postfix hexadecimal coding for a single-space carriage control (carriage-return/line-feed combination, print buffer contents, return) is obtained by placing the value 1 in the second (prefix) byte and the sum of the bit 7 value (80) and the return value (D) in the third postfix byte.

```
 80  (bit 7 = 1)
+ D  (return)
----
 8D  (postfix = return)
```

Table 5–11 Write Function Carriage Control (P4 byte 0 = 0)

Prefix/Postfix Bytes (Hexadecimal)				
Bit 7		**Bits 0–6**		**Meaning**
0		0		No carriage control is specified (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	0–1F	Output the single ASCII control character specified by the configuration of bits 0 through 4 (7-bit character set).
1	1	0	0–1F	Output the single ASCII control character specified by the configuration of bits 0 through 4, which are translated as ASCII characters 128 through 159 (8-bit character set; see Appendix B).
1	1	1	0–1F	Reserved.

5.4.3 Set Mode

Set mode operations affect the operation and characteristics of the associated terminal line. The operating system provides two types of set mode functions: set mode and set characteristics.

The set mode function affects the mode and temporary characteristics of the associated terminal line. Set mode is a logical I/O function and requires no privilege. (If you do not have LOG_IO or PHY_IO privilege, the terminal driver does not accept a set mode request to a terminal that does not have the extended terminal characteristic TT2$M_SETSPEED—even if no request for a change of speed is made. Privilege is not required if TT2$M_SETSPEED is set but no attempt to change the speed is made.) The following function code is provided:

IO$_SETMODE

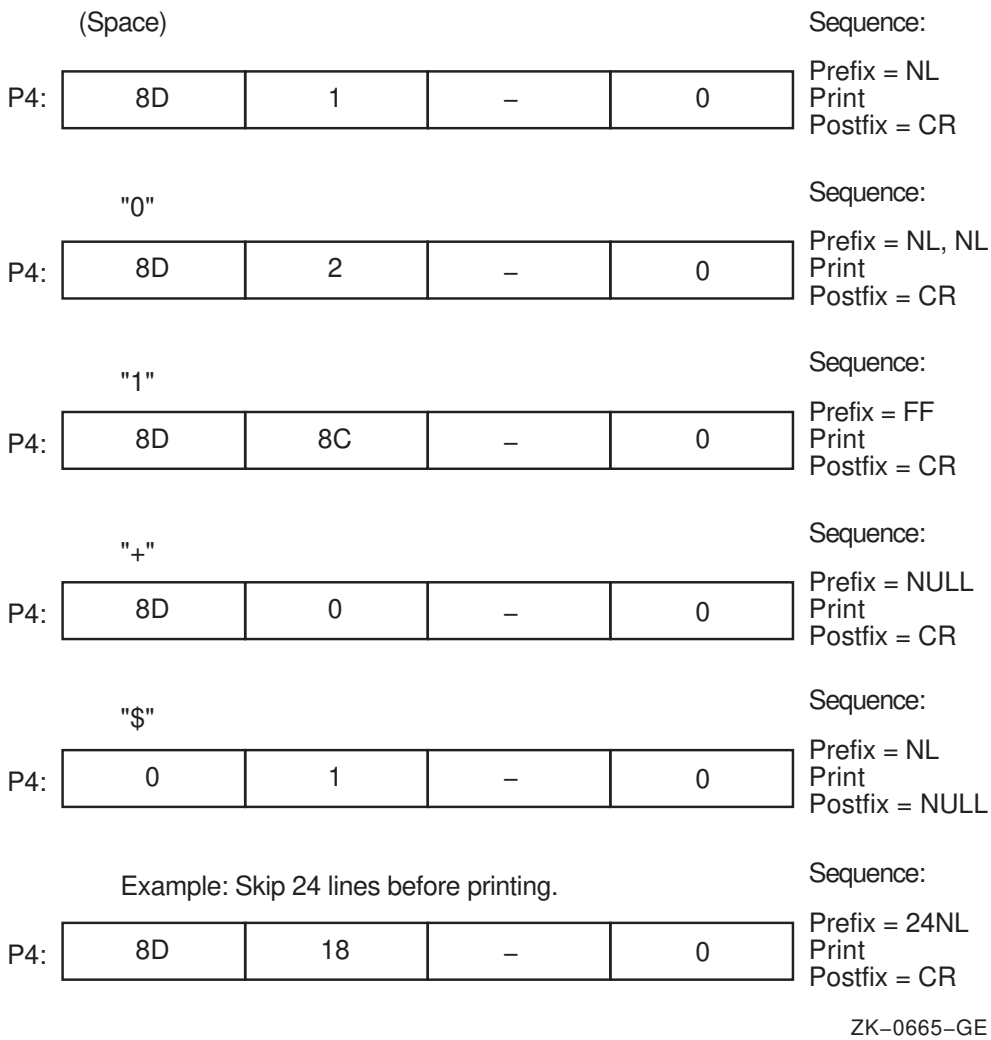
The set characteristics function affects the permanent characteristics of the associated terminal line. Set characteristics is a physical I/O function and requires the privilege necessary to perform physical I/O. The following function code is provided:

IO$_SETCHAR

The set mode and set characteristics functions take the following device- or function-dependent arguments if no function modifiers are specified:

- P1—Address of characteristics buffer
- P2—Length of characteristics buffer (default length is 8 bytes)
- P3—Speed specifier (bits 0 through 7 = transmit; 8 through 15 = receive)
- P4—Fill specifier (bits 0 through 7 = CR fill count; bits 8 through 15 = LF fill count)

Figure 5–6 Write Function Carriage Control (Prefix and Postfix Coding)

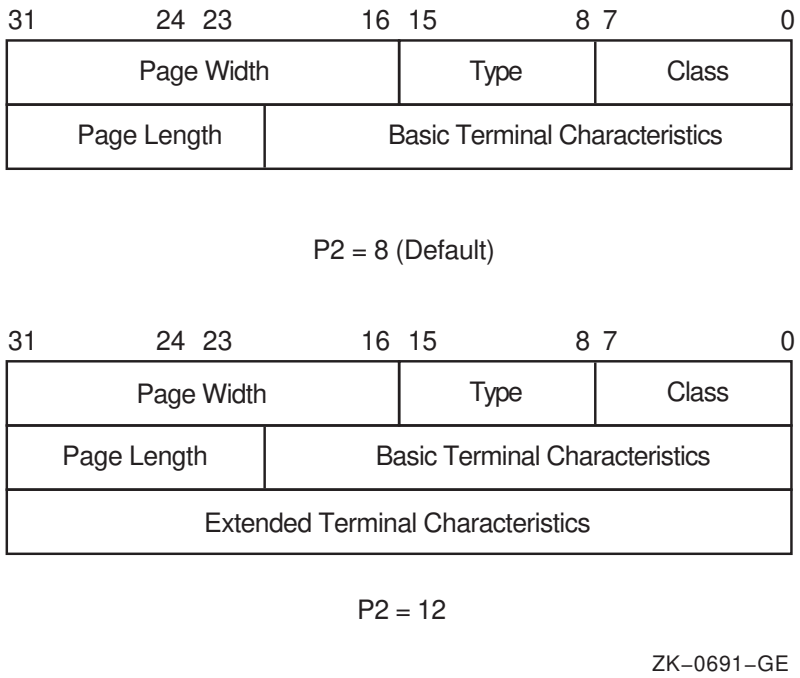


- P5—Parity flags

The P1 argument points to a variable length block, as shown in Figure 5–7. With the exception of terminal characteristics, the contents of the block are the same for both the set mode and set characteristics functions.

In the buffer, the device class is DC$_TERM, which is defined by the $DCDEF macro. The terminal type is defined by the $TTDEF macro, for example, TT$_ LA36. The page width is a value in the range of 1 through 511. The page length is a value in the range of 0 through 255. Table 5–5 lists the values for terminal characteristics. Table 5–6 lists the extended terminal characteristics. Characteristics values are defined by the $TTDEF and $TT2DEF macros.

Figure 5–7 Set Mode and Set Characteristics Buffers



Note

Make sure that the selected device is a terminal before performing any set mode function, particularly when using SYS$INPUT or SYS$OUTPUT.

The P3 argument defines the device speed, such as TT$C_BAUD_300. The low eight bits specify the transmit speed, and the high eight bits specify the receive speed. If no receive speed is specified, the indicated transmit speed is used for both transmitting and receiving. If neither the transmit nor the receive speed is specified (P3 = 0), the baud rate is not changed. The terminal driver ignores the receive speed bits for interfaces that do not support split-speed operation. While speeds up to 19.2K baud can be specified, not all controllers support all speed combinations. Refer to the associated hardware documentation to determine which speeds are supported by your controller.

P4 contains fill counts for the carriage-return and line-feed characters. Bits 0 through 7 specify the number of fill characters used after a carriage return. Bits 8 through 15 specify the number of fill characters used after a line feed.

P4 is applicable only if TT$M_CRFILL or TT$M_LFFILL is specified as a terminal characteristic for the current QIO request; see Table 5–5.

Several parity flags can be specified in the P5 argument:

- TT$M_ALTRPAR—Alter parity. If set, check the state of TT$M_PARITY and TT$M_ODD and, if indicated, change the parity. Otherwise, ignore these bits.
- TT$M_PARITY—Enable parity on terminal line if set, disable if clear.
- TT$M_ODD—Parity is odd if set.
- TT$M_ALTDISPAR—Alter dismiss parity errors. If set, check the state of TT$M_DISPARERR.

- TT$M_DISPARERR—Dismiss parity errors. If this mode is set a character with a parity error is passed to the reader. An error message is not reported.

Note

If parity is enabled, the DZ11 generates a parity check bit to detect parity mismatch. Unless TT$M_DISPARERR is enabled, parity errors that occur during an I/O read operation are fatal to the operation. Parity errors that occur on input characters (that is, keys pressed on the keyboard) when no I/O operation is in progress might result in a character loss.

- TT$M_BREAK—Generate a break if set. The break is in effect until this bit is turned off. TT$M_BREAK is supported by the LTDRIVER for terminal servers that support the break capability, such as the DECserver 200 and DECserver 500. However, in the case of LAT terminals, the terminal server controls the duration of the break.
- TT$M_ALTFRAME—If set, the four low-order bits of P5 become the frame size. Note that the frame size is for data bits only and is exclusive of parity. TT$M_ALTFRAME is supported for frame sizes of 7 and 8 for LAT devices.

To take the existing parity settings, modify them, and use them in the set mode or set characteristic function, move the byte starting at the second nibble of the buffer that is going to be used in the P5 argument. For example, the following instructions change the parity from even to odd:

```
insv    iosb+6, #4, #8, flags
bisl    #tt$m_altrpar!tt$m_odd!tt$m_parity, flags
```

The following instruction then resets the parity to its original state:

```
bicl    #tt$m_odd!tt$m_parity, flags
```

See Section 5.2.5 for information about the SET TERMINAL/FRAME command.

Application programs that change terminal 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.
4. If the characteristic is intended to be reset when the image exits, the application must perform this operation.

Failure to follow this sequence will result in clearing any previously set characteristic.

Two stop bits are used only for data rates less than or equal to 150 baud; higher data rates default to one stop bit.

The set mode and set characteristics functions can take the enable Ctrl/C AST, enable Ctrl/Y AST, enable out-of-band AST, hangup, set modem, broadcast, and loopback function modifiers that are described in the next several sections.

Note

If an attempt is made to turn on TT2$V_FALLBACK for a disconnected virtual terminal (_VTAx:) or if the Terminal Fallback facility has not been

activated, the status code SS$_BADPARAM will be returned. For more information on TFF, refer to the *OpenVMS Terminal Fallback Utility Manual*.

5.4.3.1 Hangup Function Modifier

The hangup function disconnects a terminal that is on a dialup line. (Dialup lines are described in Section 5.2.3.) The following combinations of function code and modifier are provided:

- IO$_SETMODE!IO$M_HANGUP
- IO$_SETCHAR!IO$M_HANGUP

The hangup function modifier takes no arguments. SS$_NORMAL is returned in the I/O status block.

Note

For remote terminals, the hangup function breaks the network connection to the local system ending the remote terminal session.

5.4.3.2 Enable Ctrl/C AST and Enable Ctrl/Y AST Function Modifiers

Both set mode functions can take the enable Ctrl/C AST and enable Ctrl/Y AST function modifiers. These function modifiers request the terminal driver to queue an AST for the requesting process when you press Ctrl/C or Ctrl/Y. The following combinations of function code and modifier are provided:

- IO$_SETMODE!IO$M_CTRLCAST—Enable Ctrl/C AST
- IO$_SETMODE!IO$M_CTRLYAST—Enable Ctrl/Y AST

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

P1—Address of the AST service or 0 if the corresponding AST is disabled
P2—AST parameter
P3—Access mode to deliver AST (maximized with caller's access mode)

If the respective enabling is in effect, pressing Ctrl/C or Ctrl/Y gains the attention of the enabling process (see Table 5–2).

Enable Ctrl/C and Ctrl/Y AST are one-time enabling function modifiers. After the AST occurs, it must be explicitly reenabled by one of the two function code combinations before an AST can occur again. This function code is also used to disable the AST. The function is subject to AST quotas.

You can have more than one Ctrl/C or Ctrl/Y enabled; pressing Ctrl/C, for example, results in the delivery of all Ctrl/C ASTs. ASTs are queued and delivered to the user process on a first-in/first-out basis for each access mode. However, ASTs are processed in the reverse order of the Ctrl/C AST or Ctrl/Y AST requests that have been issued to the terminal driver (on a last-in/first-out basis).

If no enable Ctrl/C AST is present, the holder of an enable Ctrl/Y AST receives an AST when Ctrl/C is pressed; carriage-return/line-feed combination, ^Y, and Return are echoed.

Figure 5–8 shows the relationship of Ctrl/C and Ctrl/Y with the out-of-band function. If Ctrl/C or Ctrl/Y is an enabled out-of-band character, any out-of-band ASTs specified for this character are delivered. If IO$M_INCLUDE function modifier is included in the out-of-band AST request for this character, an enabled Ctrl/C or Ctrl/Y AST is also delivered.

Enable Ctrl/C AST requests are flushed by the Cancel I/O on the Channel ($CANCEL) system service. Enable Ctrl/Y AST requests are flushed by the Deassign I/O Channel ($DASSGN) system service.

Ctrl/Y is normally used to gain the attention of the command interpreter and to input special commands such as DEBUG, STOP, and CONTINUE. Programs that are run from a command interpreter should not enable Ctrl/Y. Because ASTs are delivered on a first-in/first-out basis, the command interpreter's AST routine gets control first, and might not allow the program's AST to be delivered at all. Programs that require the use of Ctrl/Y should use the LIB$DISABLE_CTRL RTL routine to disable DCL recognition of Ctrl/Y.

Section 5.2.1.2 describes other effects of Ctrl/C and Ctrl/Y.

5.4.3.3 Set Modem Function Modifier

The set modem function modifier is used in maintenance operations to allow a process to activate and deactivate modem control signals. Both set mode and set characteristics functions can take the set modem function modifier. The following combinations of function code and modifier are provided:

- IO$_SETMODE!IO$M_SET_MODEM!IO$M_MAINT
- IO$_SETCHAR!IO$M_SET_MODEM!IO$M_MAINT

Note

For LAT devices, the set modem field for maintenance operations of the IO$M_SET_MODEM!IO$M_MAINT function modifier is unsupported and may return unpredictable results.

These function code modifier pairs take the following device- or function-dependent argument:

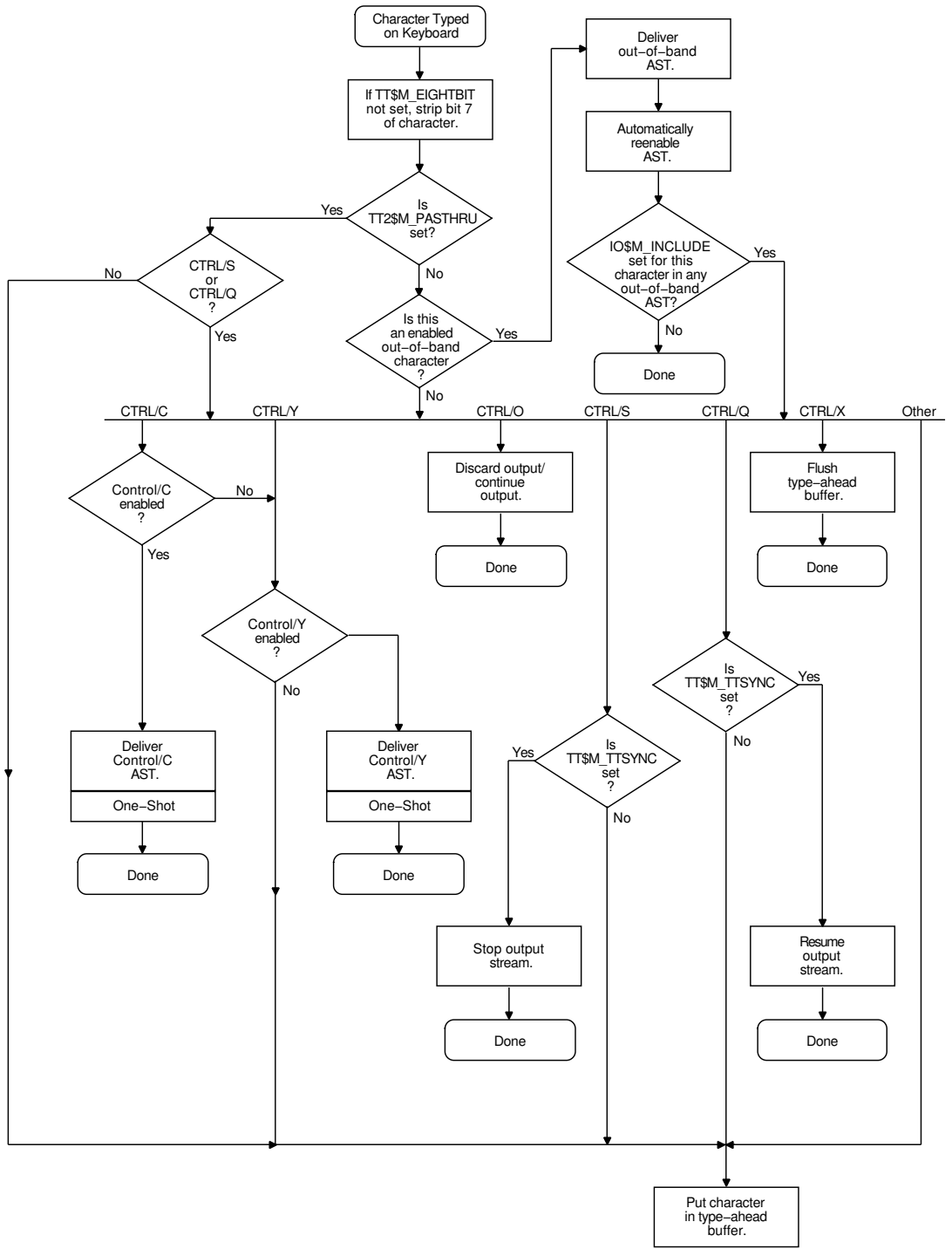
P1—The address of a quadword block that specifies which modem control signals to activate or deactivate

Figure 5–9 shows the format of this block.

The modem on and modem off fields, in combination or separately, can specify one or more of the following values:

- TT$M_DS_RTS—Request to send (RTS)
- TT$M_DS_DTR—Data terminal ready (DTR)
- TT$M_DS_SECTX—Transmitted backward channel data (Sec Txd)

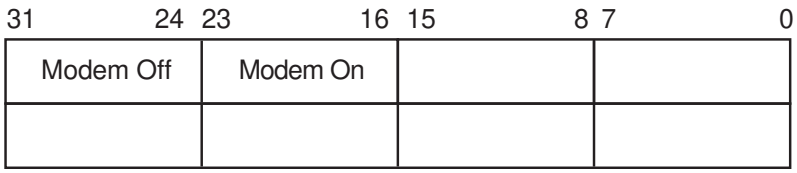
Figure 5–8 Relationship of Out-of-Band Function with Control Characters



ZK–1202–GE

The $TTDEF macro defines the values for these values. These values can only be specified if the terminal characteristic TT$M_MODEM is not set. Otherwise, an error (SS$_ABORT) will result.

Figure 5–9 Set Mode P1 Block



ZK–0692–GE

Note 1

The set modem function is not supported for remote terminals. The status SS$_DEVREQERR is returned in the I/O status block.

Note 2

Because the DMF32 does not provide the secondary transmitted data signal (Sec Txd), the driver sets the secondary request to send the signal. Users should connect a jumper cable between pins 14 and 19 on the DMF32.

5.4.3.4 Loopback Function Modifier

The loopback function modifier is used in maintenance operations to place the terminal line in a hardware loopback mode. Data transmitted to a line in this mode is returned as receive data. If the controller does not support loopback mode or the terminal line has the TT$M_MODEM characteristic set, an error status (SS$_ABORT) is returned. Both set mode functions can take the loopback function modifier.

Note

The loopback function is not supported for remote terminals. The status SS$_DEVREQERR is returned in the I/O status block.

The following combinations of function code and modifier are provided:

- IO$_SETMODE!IO$M_LOOP!IO$M_MAINT
- IO$_SETCHAR!IO$M_LOOP!IO$M_MAINT

Data transmitted in the loopback mode should only be written in records less than or equal to the size of the type-ahead buffer (see Section 5.2.1.5). Programs that use the loopback function modifier should incorporate a 1-second delay to allow the controller to enable the loopback mode after the request is posted. Write requests should also include the IO$M_NOFORMAT function modifier to prevent terminal driver from formatting input or output data.

Note

The serial line interfaces for the VAX 8200 processor implement an internal loopback bus that is common to all four serial lines. The hardware allows all serial lines operating in loopback mode to transmit data to the bus at the same time. If more than one serial line writes data

to the bus, all of the transmitted data is combined and made available to the receiving end of those same serial lines. Thus, the received data may be different from the transmitted data if more than one serial line is operating in loopback mode at the same time. To prevent receiving such spurious data, you must must not operate multiple serial lines in loopback mode.

The operating system provides another function modifier to reset a terminal line previously placed in loopback mode. The following combinations of function code and modifier are provided:

- IO$_SETMODE!IO$M_UNLOOP!IO$M_MAINT
- IO$_SETCHAR!IO$M_UNLOOP!IO$M_MAINT

Programs that use the unloop function modifier should incorporate a 1-second delay to allow the controller to reset the loopback mode after the request is posted.

Note

IO$M_LOOP and IO$M_UNLOOP are not supported for LAT devices.

5.4.3.5 Enable Out-of-Band AST Function Modifier

The enable out-of-band AST function modifier requests that the terminal driver queue an AST for the requesting process when you enter any one of 32 control characters. The following combinations of function code and modifier are provided:

- IO$_SETMODE!IO$M_OUTBAND—Enable out-of-band AST
- IO$_SETCHAR!IO$M_OUTBAND—Enable out-of-band AST

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

- P1—Address of the AST service or 0 if the AST entered on this channel is to be canceled. (The AST parameter will be the out-of-band character.)
- P2—Address of a character mask with the same format as the short form terminator mask (see Section 5.4.1.2).
- P3—Access mode to deliver AST (maximized with the caller's access mode).

The IO$_SETMODE!IO$M_OUTBAND function can optionally take the following function modifiers:

- IO$M_INCLUDE—Include the character typed in the data stream.
- IO$M_TT_ABORT—Allow current read and write operations to be aborted. (The IOSB for aborted operations returns the status SS$_CONTROLC.)

If an out-of-band AST is in effect, pressing any control character specified in the P2 mask gains the attention of the enabling process. Figure 5–8 shows the relationship of the out-of-band function with some of the control characters.

You can have only one out-of-band AST enabled per channel.

Out-of-band ASTs are repeating ASTs; they continue to be delivered until specifically disabled. Out-of-band AST enables are flushed by the Cancel I/O on Channel ($CANCEL) system service.

5.4.3.6 Broadcast Function Modifier

The broadcast function modifier allows you to turn on or turn off selected broadcast requester identifiers (IDs). The following combination of function code and modifier is provided:

IO$_SETMODE!IO$M_BRDCST

This function code modifier pair takes the following device- or function-dependent arguments:

- P1—A buffer that contains the bits that specify the requester IDs to be broadcast
- P2—The length of the P1 buffer (default is 8 bytes)

The first longword of P1 is reserved for use by Digital facilities, as shown in Table 5–12. The symbols are defined in the system macro library ($BRKDEF). The second longword is for customer use to specify selected bits. If any bit is set in the P1 buffer, that particular requester ID is turned off for broadcast.

Table 5–12 Broadcast Requester IDs

Bit	Meaning
BRK$C_DCL	Disables broadcasts by Ctrl/T
BRK$C_GENERAL	Disables broadcasts by the DCL command REPLY and the SYS$BRDCST system service
BRK$C_MAIL	Disables broadcasts by the Mail utility
BRK$C_PHONE	Disables broadcasts by the Phone utility
BRC$C_QUEUE	Disables broadcasts about batch and print queues
BRK$C_SHUTDOWN	Disables broadcasts about system shutdown
BRK$C_URGENT	Disables broadcasts labeled URGENT by the REPLY command
BRK$C_USER*n*	Disables broadcasts by images associated with the specified value; *n* can be any decimal integer between 1 and 16

5.4.4 LAT Port Driver QIO Interface

The LAT port driver (LTDRIVER) accommodates I/O requests from application programs for connections to remote devices on one or more terminal servers; for connections to remote services; and for configuring LTDRIVER and retrieving configuration information about LTDRIVER. A remote device, such as a printer, can be shared in a LAT configuration. Before an application program can access a remote device, the system manager must create logical devices and map them to physical devices connected to terminal servers. Creating and mapping these logical devices can be done either with the LAT Control Program (LATCP) utility or with a $QIO request from a program that has OPER privilege. Once mapped, application programs can establish and terminate connections to these remote devices.

This section describes the capabilities of the QIO interface to the LAT port driver (LTDRIVER). The QIO interface allows application programs to access and modify information contained in the LTDRIVER data structures and to initiate events and obtain status information. You must use these QIO functions to establish a connection to a remote device or service from an application program. Digital does not support any other methods of connection.

The LTDRIVER responds to TEST SERVICE commands issued at terminal servers that support the TEST SERVICE command, such as the DECserver 200 and DECserver 500 servers.

LAT devices can use all read and write function modifiers listed for the terminal driver function codes except those modifiers that apply to modems (see Sections 5.4.1 and 5.4.2).

The operating system does not support the following set mode or set characteristics function code modifiers for LAT devices:

- IO$M_LOOP
- IO$M_UNLOOP
- TT$M_ALTRPAR
- TT$M_ALTFRAME
- TT$M_MODEM
- TT$M_READSYNC
- TT2$M_SETSPEED

With LAT devices, the terminal server, rather than the host, handles flow control to the physical device. A separate flow control mechanism exists between the server and the host.

5.4.4.1 LAT Port Types

QIO functions can be used to create the following LAT port types:

- Application Port. This type of port can be used to connect to a remote device (typically a printer) on a terminal server or to a dedicated port on another LAT service node. This is the default port type. See Section 5.4.4.5 for a description of programming an application port.
- Dedicated Port. This type of port specifies that the logical port on your node is dedicated to an application service. When users on a terminal server (or on another node that supports outgoing connections) request a connection to this service name, they are connected to a dedicated port. See Section 5.4.4.6 for a description of programming a dedicated port and application service.
- Forward Port. This type of port is used for outgoing LAT connections (to remote services) and is created by assigning a channel to the LAT template device _LTA0: with the $ASSIGN system service.

  QIO functions can also be used to configure and read information about these ports; for more information:

  - See Section 5.4.4.3 for a description of configuring a LAT port
  - See Section 5.4.4.4 for a description of reading configuration information about a LAT port

- See Section 5.4.4.7 for a description of programming a forward port in order to make a connection to a LAT service

5.4.4.2 LAT Port Driver Functions

The operating system provides the following combinations of function code and modifier:

- IO$_TTY_PORT!IO$M_LT_CONNECT. Requests that the LAT port driver make a connection to a remote device on a server (or dedicated port on another LAT service node) or to a remote service, depending on whether the port is an application port or a forward port respectively. For dedicated ports, this QIO completes when an incoming connection to the port is established. See Section 5.4.4.5 for a description of programming an application port, Section 5.4.4.6 for a description of programming a dedicated port, and Section 5.4.4.7 for a description of programming a forward port.
- IO$_TTY_PORT!IO$M_LT_DISCON. Depending on the port type, requests that the LAT port driver terminate the LAT connection to the remote device, service, or local application service. IO$M_FLUSH_FLAG can be specified in the P2 argument to IO$M_LT_DISCON. The flush flag indicates that any data not delivered to the remote device is to be flushed when the disconnect is issued.
- IO$_TTY_PORT!IO$M_LT_SETMODE. Requests that the LAT port driver create or configure a LAT entity. See Section 5.4.4.3 for more information.
- IO$_TTY_PORT!IO$M_LT_SENSEMODE. Requests that the LAT port driver return configuration information about a LAT entity. See Section 5.4.4.4 for more information.

5.4.4.3 Creating and Configuring LAT Entities

The LAT SETMODE $QIO function (IO$_TTY_PORT!IO$M_LT_SETMODE) is used to create, delete, and modify LAT nodes, services, ports, and links.

Creation, deletion, or modification of any entity requires the OPER privilege.

The LAT SETMODE $QIO function accepts four arguments: P1, P2, P3, and P4. P1 is the address of an item list; P2 is the length of this item list.

P3 specifies the type of entity to which the SETMODE operation applies. The entity type can be one of five types:

- Node (LAT$C_ENT_NODE). Only the local node name may be specified, with the exception of a SETMODE itemlist containing no item codes other than LAT$_ITM_COUNTERS.
- Service (LAT$C_ENT_SERVICE). Only local service names may be specified, with the exception of a SETMODE itemlist containing no item codes other than LAT$_ITM_COUNTERS.
- Link (LAT$C_ENT_LINK). The data link associated with the LAN.
- Port (LAT$C_ENT_PORT).

Alpha

- Queue Entry (LAT$C_ENT_QUEUE_ENTRY). Indicates queue entry entities. When this entity is used, the only valid SETMODE operation is delete. ♦

The value for the entity type occupies the low-order 16 bits (bits 0–15) of the P3 parameter. For all four entity types, bits 16–19 are used as a status field to indicate the expected current status of the entity. These bits are used to decide whether the entity needs to be created before its characteristics are set. The possible values for this field are:

- LAT$C_ENTS_OLD—The entity must already exist. An SS$_NOSUCHDEV error is returned if the entity does not exist.
- LAT$C_ENTS_NEW—The entity must be created. An SS$_DUPLNAM error is returned if the entity already exists.
- LAT$C_ENTS_UNK—If the entity does not exist, it is created. If it does exist, its characteristics are modified.
- LAT$C_ENTS_DEL—If the entity exists, delete it. Otherwise, an SS$_ NOSUCHDEV error is returned and the item list is not used.

P4 may contain the address of an entity name string descriptor. If this parameter is omitted (contains a 0 or the address of a descriptor that points to an empty buffer), a default may be used in some cases. The defaults for each entity type are as follows:

- LAT$C_ENT_NODE—The local node.
- LAT$C_ENT_SERVICE—No default; you must specify the service name.
- LAT$C_ENT_LINK—The string LAT$LINK.
- LAT$C_ENT_PORT—The device name associated with the currently assigned channel (the CHAN parameter of the $QIO function).

SETMODE can return the following status codes:

- SS$_NOPRIV—No privilege to complete the desired operation.
- SS$_ACCVIO—Part of the argument list or itemlist is not addressable.
- SS$_BADPARAM—One of the parameters in the itemlist is in error. If this value is returned, the second longword of the IOSB contains the item code of the parameter in error.

SETMODE Item Codes

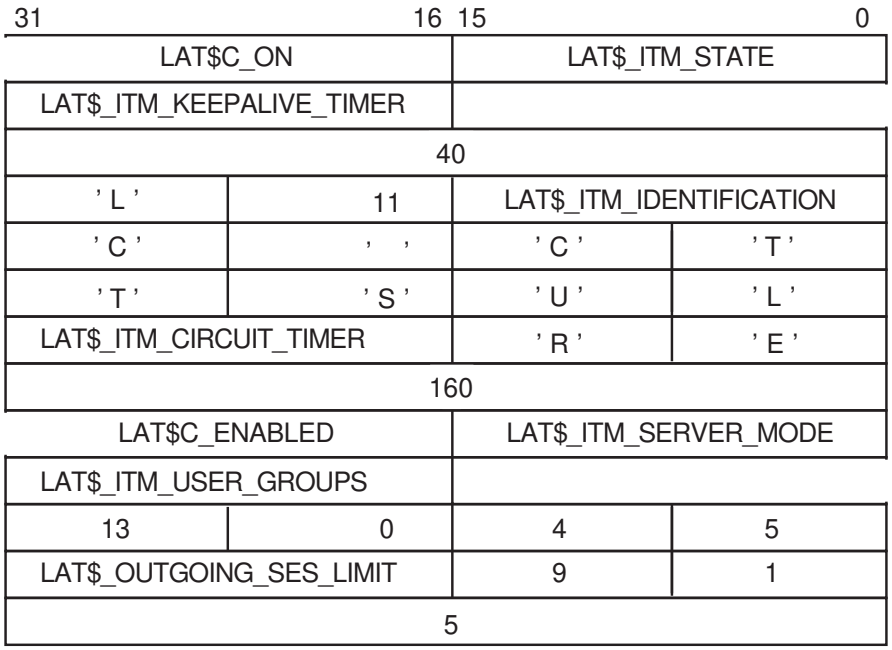
Each item in the itemlist consists of a 1-word (16-bit) item code, followed by a value associated with the item.

Item codes in which the bit named LAT$V_STRING is zero take a longword value. The associated value is contained in the longword immediately following the item code in the itemlist. Item codes in which this bit is 1 take a counted string for their value. The byte immediately following the item code contains a byte count, which describes the length of the string that immediately follows it.

If you set bit LAT$V_CLEAR in the item code to 1, the current value associated with the item code is cleared or set to its default value. In this case, the actual value specified in the itemlist is ignored, although the byte count field skips to the next item in the itemlist.

Figure 5–10 shows an example of a SETMODE itemlist.

Figure 5–10 Example SETMODE Itemlist



ZK–3798A

This SETMODE itemlist is the P1 parameter for a $QIO SETMODE function on the local node. P4 is omitted, and P3 is #LAT$C_ENT_NODE!<LAT$C_ENTS_OLD@16>. P2 is the length of the itemlist (52). A $QIO SETMODE function for this itemlist would perform the following operations:

1. Set the state of the node to on.
2. Set the LAT keepalive timer to 40 seconds.
3. Set the node identification to LTC CLUSTER.
4. Set the LAT circuit timer to 160 milliseconds.
5. Enable LAT outbound connections.
6. Turn on user groups 2, 8, 10, 11, 12, 16, and 19. LAT$_ITM_USER_GROUPS is represented by a bit field.
7. Set the outgoing session limit to five sessions.

For each entity type, only a subset of item codes may be set. Table 5–13 lists the item codes that may be set for the LAT$C_ENT_NODE entity type.

Table 5–13 LAT$C_ENT_NODE Item Codes

Item Code	Meaning	
LAT$_ITM_STATE	Operating state of the LAT protocol. The following values are allowed:	
	LAT$C_OFF	Turn off LAT protocol processing. No new connections allowed in either direction. Existing connections are terminated immediately. This is the default.
	LAT$C_SHUT	Disallow new LAT connections in either direction. Existing connections are allowed to remain active.
	LAT$C_ON	Turn on LAT protocol processing.
LAT$_ITM_CIRCUIT_ TIMER	Circuit timer value in milliseconds. Valid values are 10 to 1000 milliseconds. The default is 80 milliseconds.	
LAT$_ITM_CPU_RATING	CPU rating. Valid values are 0 to 100. If this value is 0, then the CPU rating value is not used in the rating calculation. See the *OpenVMS System Management Utilities Reference Manual* for a complete description of this feature.	
LAT$_ITM_DEVICE_SEED	Overrides the default lower boundary for new LTA devices. Valid values are 0 to 9999; the default is 0. See the *OpenVMS System Management Utilities Reference Manual* for more information on this feature.	
LAT$_ITM_KEEPALIVE_ TIMER	Keepalive timer value in seconds. Valid values are 10 to 255 seconds. The default is 20 seconds.	
LAT$_ITM_MULTICAST_ TIMER	Multicast timer value in seconds. Valid values are 10 to 180 seconds. The default is 60 seconds.	
LAT$_ITM_NODE_LIMIT	Maximum number of nodes in LAT database. The default is zero, where the maximum is determined by system resources.	
LAT$_ITM_RETRANSMIT_ LIMIT	LAT retransmit limit. Valid values are 4 to 120 retransmissions. The default is eight retransmissions.	
LAT$_ITM_SERVER_ MODE	Controls whether the node allows the use of the MASTER side of the LAT protocol for outbound connections. Valid values are:	
	LAT$C_DISABLED	Server mode disabled (this is the default).
	LAT$C_ENABLED	Server mode enabled.
LAT$_ITM_SERVICE_ RESPONDER	Indicates whether the node is to respond to service inquiries originating from a remote system. These inquiries are not necessarily directed at services being offered by the node. See the *OpenVMS System Management Utilities Reference Manual* for a complete description of this feature. Valid values are:	
	LAT$C_DISABLED	Service responder disabled (this is the default).
	LAT$C_ENABLED	Service responsder enabled.
LAT$_ITM_OUTGOING_ SES_LIMIT	Maximum number of outgoing LAT sessions. A value of zero, which is the default, indicates that the limit is determined by system resources.	
LAT$_ITM_INCOMING_ SES_LIMIT	Maximum number of interactive LAT sessions. A value of zero, which is the default, indicates that the limit is determined by system resources.	
LAT$_ITM_ CONNECTIONS	Controls whether inbound connections can be accepted. Valid values are:	
	LAT$C_DISABLED	Inbound connections disabled.
	LAT$C_ENABLED	Inbound connections enabled (this is the default).
LAT$_ITM_NODE_NAME	Causes the LAT node name to be set to the given name. This item code may be specified only if the entity status field of the P3 parameter is LAT$C_ ENTS_NEW; otherwise, a LAT$_ENTNOTFOU error results.	

(continued on next page)

Table 5–13 (Cont.) LAT$C_ENT_NODE Item Codes

Item Code	Meaning
LAT$_ITM_ IDENTIFICATION	Node identification string. The default is the translation of SYS$ANNOUNCE.
LAT$_ITM_SERVICE_ GROUPS	Specifies a default service group code bit mask. This mask is then used when creating new local services. The default is group code 0 enabled and all others disabled when the LAT software is initialized. Note that the use of the LAT$V_CLEAR bit is an exception for this parameter code. If you clear bit LAT$V_CLEAR, group codes corresponding to the group code mask, as specified in the itemlist, are set. Alternatively, if you set LAT$V_CLEAR, group codes corresponding to the group code mask, as specified in the itemlist, are cleared.
LAT$_ITM_USER_ GROUPS	LAT group codes to be used when attempting outbound connections using the MASTER side of the LAT protocol. The default is all group codes disabled when the LAT software is initialized. Note that the use of the LAT$V_CLEAR bit is an exception for this parameter code. If you clear bit LAT$V_CLEAR, group codes corresponding to the group code mask, as specified in the itemlist, are set. Alternatively, if you set LAT$V_CLEAR, group codes corresponding to the group code mask, as specified in the itemlist, are cleared.
LAT$_ITM_COUNTERS	Node counters block. Allows for zeroing of all node counters. This item code may be specified only if the entity status field of the P3 parameter is LAT$C_ENTS_OLD and the LAT$V_CLEAR bit is set. Violating either of these two rules results in a returned status of SS$_BADPARAM.
LAT$_ITM_MAXIMUM_ UNITS	Maximum unit number. Sets the highest value for a LTA unit number. Must be between 1 and 9999; defaults to 9999.
‡ LAT$_ITM_HI_CIRCUITS	Indicates the highest number the resource attained since the host was initialized for LAT connections to node.
‡ LAT$_ITM_CUR_ CIRCUITS	Indicates current count of active connections to node.
‡ LAT$_ITM_MAX_ CIRCUITS	Indicates maximum allowed virtual circuits to node.
‡ LAT$_ITM_HI_ SESSIONS	Indicates highest number the resource attained since the host was initialized for LAT sessions.
‡ LAT$_ITM_CUR_ SESSIONS	Indicates current number of active sessions.
‡ LAT$_ITM_MAX_ SESSIONS	Indicates maximum possible sessions.
‡ LAT$_ITM_HI_OUT_ QUEUE	Indicates highest number the resource attained since the host was initialized of outgoing queued connect requests.
‡ LAT$_ITM_CUR_OUT_ QUEUE	Indicates current count of outgoing queued connect requests.
‡ LAT$_ITM_MAX_OUT_ QUEUE	Indicates maximum number of simultaneous outgoing queued connect requests.
‡ LAT$_TIM_HI_IN_ QUEUE	Indicates highest number the resource attained since the host was initialized of incoming queued requests.
‡ LAT$_ITM_CUR_IN_ QUEUE	Indicates current number of entries in the incoming connect queue.

‡Alpha specific.

(continued on next page)

Table 5–13 (Cont.) LAT$C_ENT_NODE Item Codes

Item Code	Meaning
‡ LAT$_ITM_MAX_IN_QUEUE	Indicates maximum number of entries allowed on the incoming connect queue.
‡ LAT$_ITM_HI_SAMS_QUEUED	Indicates highest number the resource attained since the host was initialized of outstanding, unprocessed service announcement messages by LATACP.
‡ LAT$_ITM_CUR_SAMS_QUEUED	Indicates current number of outstanding, unprocessed service announcement messages on LATACP's queue.
‡ LAT$_ITM_MAX_SAMS_QUEUED	Indicates maximum number of outstanding, unprocessed service announcement messages allowed on LATACP's queue. If this limit is ever reached, subsequent service announcement messages are not delivered or processed by LATACP.
‡ LAT$_ITM_HI_SOL_QUEUED	Indicates highest number the resource attained since the host was initialized of outstanding, unprocessed solicit information messages by LATACP.
‡ LAT$_ITM_CUR_SOL_QUEUED	Indicates current number of outstanding, unprocessed solicit information messages on LATACP's queue.
‡ LAT$_ITM_MAX_SOL_QUEUED	Indicates maximum number of outstanding, unprocessed solicit information messages allowed on LATACP's queue. If this limit is ever reached, subsequent solicit information messages are not delivered or processed by LATACP.
‡ LAT$_ITM_HI_AVAIL_SVCS	Indicates highest number the resource attained since the host was initialized of number of available services in LATACP database.
‡ LAT$_ITM_CUR_AVAIL_SVCS	Indicates count of currently available LAT services in LATACP database.
‡ LAT$_ITM_MAX_AVAIL_SVCS	Indicates maximum number of available services possible in LATACP database.
‡ LAT$_ITM_HI_REACH_NODES	Indicates highest number the resource attained since the host was initialized of reachable nodes in LATACP database.
‡ LAT$_ITM_CUR_REACH_NODES	Indicates current number of reachable nodes in LATACP database.
‡ LAT$_ITM_MAX_REACH_NODES	Indicates maximum number of nodes allowed in LATACP database.
‡ LAT$_ITM_HI_LCL_SVCS	Indicates highest number the resource attained since the host was initialized of locally offered services.
‡ LAT$_ITM_CUR_LCL_SVCS	Indicates current count of locally offered service.
‡ LAT$_ITM_MAX_LCL_SVCS	Indicates maximum number of locally offered services.
‡ LAT$_ITM_DISCARDED_NODES	Indicates number of discarded service announcement messages.
‡ LAT$_ITM_SERVICE_CLASSES	Indicates returned service class bit mask for supported service classes on node. It is returned for both local and remote nodes. If service class 1 is enabled, then bit <1> is set in this mask. When bit setting equals 1, this indicates the corresponding service class for that bit is enabled. That is, when bit <3> equal 1, then service class 3 is enabled.
LAT$_ITM_LARGE_BUFFERS	Indicates in Boolean logic whether or not the LAT software is using large packet support by default.
LAT$_ITM_ANNOUNCEMENTS	Indicates in Boolean logic whether or not the LAT software is transmitting LAT service advertisement messages.

‡Alpha specific.

Table 5–14 lists the item codes that may be set for the LAT$C_ENT_SERVICE entity type.

Table 5–14 LAT$C_ENT_SERVICE Item Codes

Item Code	Meaning	
LAT$_ITM_RATING	Static LAT service rating. The default is the dynamic rating calculation. Static ratings can be between 0 and 255.	
LAT$_ITM_ IDENTIFICATION	Service identification string. The default is the translation of SYS$ANNOUNCE.	
LAT$_ITM_SERVICE_ TYPE	Defines the type of service. Valid values are:	
	LAT$C_ST_GENERAL	Creates a general timesharing service.
	LAT$C_ST_APPLICATION	Creates a special application service that must then be associated with ports dedicated to accepting connections to this service (dedicated ports).
	‡ LAT$C_ST_LIMITED	Indicates that the service is limited.
LAT$_ITM_COUNTERS	Service counters block. Allows for zeroing of all service counters. This item code may be specified only if the entity status field is LAT$C_ENTS_OLD and the LAT$V_CLEAR bit is set. Violating either of these two rules results in a returned status of SS$_BADPARAM.	
‡ LAT$_ITM_PASSWORD	Indicates that if a value of LAT$C_ENABLED is indicated, then the service is password protected. Indicates that if a value of LAT$C_DISABLED is indicated, then the service is not password protected.	
‡ LAT$_ITM_LIM_PORT_ BLOCK	Indicates a subblock contained in an itemlist, which has a list of limited ports associated with the named service. This subblock may be repeated several times; that is, once for each limited LAT device associated with the specified service.	

‡Alpha specific.

Table 5–15 lists the item codes that may be set for the LAT$C_ENT_LINK entity type.

Table 5–15 LAT$C_ENT_LINK ItemCodes

Item Code	Meaning	
LAT$_ITM_STATE	Operating state of the LAT protocol. Valid values are:	
	LAT$C_OFF	Turn off LAT protocol processing. No new connections allowed in either direction. Existing connections are terminated immediately.
	LAT$C_SHUT	Disallow new LAT connections in either direction. Existing connections are allowed to remain active.
	LAT$C_ON	Turn on LAT protocol processing. This is the default.
LAT$_ITM_DEVICE_ NAME	The name of the local area network (LAN) device to be used for this link. The default is hardware dependent.	
LAT$_ITM_DECNET_ ADDRESS	Specifies whether to use the DECnet address when starting the LAT protocol on the LAN controller associated with this link. Valid values are:	
	LAT$C_DISABLED	DECnet address use disabled.
	LAT$C_ENABLED	DECnet address use enabled (this is the default).
LAT$_ITM_COUNTERS	Link counters block. Allows for zeroing of all link counters. This item code may be specified only if the entity status field is LAT$C_ENTS_OLD and the LAT$V_CLEAR bit is set. Violating either of these two rules results in a returned status of SS$_BADPARAM.	

Table 5–16 lists the item codes that may be set for the LAT$C_ENT_PORT entity type.

Table 5–16 LAT$C_ENT_PORT Item Codes

Item Code	Meaning	
LAT$_ITM_PORT_ TYPE	Type of port. Valid values are:	
	LAT$C_PT_APPLICATION	Application port for solicited connections.
	LAT$C_PT_DEDICATED	Dedicated port associated with a local application service.
	‡ LAT$C_PT_LIMITED	Indicates that the port type is limited.
LAT$_ITM_QUEUED	Controls whether the solicited connection requests queued or nonqueued access. Valid values are:	
	LAT$C_DISABLED	Queued access disabled.
	LAT$C_ENABLED	Queued access enabled (this is the default).

‡Alpha specific.

(continued on next page)

Table 5–16 (Cont.) LAT$C_ENT_PORT Item Codes

Item Code	Meaning
LAT$_ITM_SERVICE_CLASS	Controls the class driver that the LAT driver communicates with when a connection is established. This item code can be used only with an entity status of LAT$C_ENTS_NEW. Therefore, the service class must be specified when the device is created. An attempt to change the service class of an existing device returns SS$_BADPARAM. Valid values are: LAT$C_SERVCLASS_INTERACTIVE — Service class 1, TTDRIVER (this is the default). LAT$C_SERVCLASS_XTRANSPORT — Service class 3, X Protocol. LAT$C_SERVCLASS_FONT — Service class 4, X fonts.
LAT$_ITM_DISPLAY_NUMBER	For X devices, this is the binary value of the display number, which may need to be transmitted in some LAT messages. Values range from 0–255, with a default of 0. This item code has meaning only when used with service classes 3 and 4 (LAT$C_SERVCLASS_XTRANSPORT AND LAT$C_SERVCLASS_FONT).
LAT$_ITM_TARGET_NODE_NAME	Target node name for connection. This parameter must be specified for application ports and may optionally be specified for forward ports.
LAT$_ITM_TARGET_SERVICE_NAME	Target service name for connection. This parameter must be specified for forward ports and may optionally be specified for application ports. For dedicated ports, this parameter specifies the local application service to which the port should be associated.
LAT$_ITM_TARGET_PORT_NAME	Target port name for connection. This parameter may optionally be specified for application ports or forward ports; it is ignored for all other kinds of ports.
LAT$_ITM_SERVICE_PASSWORD	Password string for remote service on forward ports. This parameter must be specified to access services that are protected with a password. This parameter is ignored if it is specified for a service that is not protected with a password.
‡ LAT$_ITM_DIALUP	Indicates if an LTA device tells a remote node that the connection is coming from a dialin source. Possible values are LAT$C_ENABLED or LAT$C_DISABLED.
‡ LAT$_ITM_AUTOPROMPT	Indicates if a connect request has autoprompt enabled. Possible values are LAT$C_ENABLED or LAT$C_DISABLED.

‡Alpha specific.

5.4.4.4 Obtaining Information About LAT Entities

The LAT SENSEMODE $QIO function (IO$_TTY_PORT!IO$M_LT_SENSEMODE) is used to obtain information about LAT nodes, services, ports, and links.

The LAT SENSEMODE $QIO function accepts four arguments: P1, P2, P3, and P4. P1 is the address of a buffer into which information about the desired entity is returned. The information is returned in the form of an item list. Unlike system services such as $GETDVI or $GETJPI, you do not select which items of information are returned. P2 is the length of the buffer specified in P1, in bytes. The number of bytes of information returned in the P1 buffer is returned in IOSB+2.

P3 specifies the type of entity to which the SENSEMODE operation applies. The entity type can be one of five types:

- Node (LAT$C_ENT_NODE). Node, including the local node.
- Service (LAT$C_ENT_SERVICE). Service, including local services.
- Link (LAT$C_ENT_LINK). Data link associated with the LAN.
- Port (LAT$C_ENT_PORT).
- Queue Entry (LAT$C_ENT_QUEUE_ENTRY). Indicates queue entry entities. ♦

Alpha

The value for the entity type occupies the low-order 16 bits (bits 0–15) of the P3 parameter. Bits 16–23 are used as a flag field. Two bits are currently defined within this field: LAT$V_SENSE_NEXT and LAT$V_SENSE_FULL. If the LAT$V_SENSE_NEXT bit is 0, information about the current entity described by the P3 and P4 parameters is returned to the user; if this bit is 1, information about the next entity that logically follows the one described by P4 is returned. If LAT$V_SENSE_FULL is 0, only those item codes marked SUMMARY in the following tables are returned; if this bit is 1, all item codes that describe the entity specified by the P3 and P4 parameters are returned.

P4 may contain the address of an entity name string descriptor. If this parameter is omitted (contains a zero or the address of a descriptor that points to an empty string) and the LAT$V_SENSE_NEXT bit is set, information about the first entity that matches the entity type supplied by P3 is returned.

If P4 is omitted and the LAT$V_SENSE_NEXT bit is 0, a default entity name may be used in some cases. The defaults for each entity type are as follows:

- LAT$C_ENT_NODE—The local node.
- LAT$C_ENT_SERVICE—No default; you must specify the service name.
- LAT$C_ENT_LINK—The string LAT$LINK.
- LAT$C_ENT_PORT—The device name associated with the currently assigned channel (the CHAN parameter of the $QIO function.)

SENSEMODE can return the following failure return codes:

- SS$_NOPRIV—No privilege to complete the desired operation
- SS$_ACCVIO—Part of the argument list or item list is not addressable

SENSEMODE Item Codes

Each item in the itemlist starts with a 1-word (16-bit) item code that describes the type of information contained in the item. The item code is followed by a value associated with the item.

Item codes in which the bit named LAT$V_STRING is 0 take a longword value. The associated value is contained in the longword immediately following the item code in the itemlist. Item codes in which this bit is 1 take a counted string for their value. The byte immediately following the item code contains a byte count, which describes the length of the string that immediately follows it.

Table 5–17 lists the item codes that are returned for the LAT$C_ENT_NODE entity type. Item codes noted as LOCAL are returned only if the information being returned is for the local node. Item codes noted as REMOTE are returned only if the information being returned is for a remote node. Item codes noted as BOTH are returned for both types of nodes.

Table 5–17 LAT$C_ENT_NODE Item Codes

Item Codes	Meaning	
LAT$_ITM_NODE_ NAME (BOTH, SUMMARY)	LAT node name for the node.	
LAT$_ITM_ IDENTIFICATION (BOTH, SUMMARY)	Node identification string.	
LAT$_ITM_NODE_ TYPE (BOTH, SUMMARY)	Type of node. Possible values are:	
	LAT$C_NT_LOCAL	Node is local node.
	LAT$C_NT_REMOTE	Node is remote node.
LAT$_ITM_STATE (LOCAL,SUMMARY)	Operating state of the LAT protocol. Possible values are:	
	LAT$C_ON	New connections are allowed and the LAT protocol is running.
	LAT$C_OFF	New connections are not allowed. The LAT protocol is not running.
	LAT$C_SHUT	No new connections are allowed. Currently active connections are still maintained. The LAT protocol remains running only until the last active session is disconnected, at which time the node is placed in the OFF state.
LAT$_ITM_NODE_ STATUS (REMOTE, SUMMARY)	Current status of remote node. This item code is present only if a LAT virtual circuit does not currently exist between the local node and this remote node. Possible values are:	
	LAT$C_REACHABLE	Remote node is reachable.
	LAT$C_UNREACHABLE	Remote node is unreachable.
	LAT$C_UNKNOWN	Remote node status is unknown.
LAT$_ITM_ CONNECTED_COUNT (REMOTE, SUMMARY)	Number of LAT sessions from the local node to this remote node. This item code replaces the LAT$_ITM_NODE_STATUS item code for remote nodes to which a LAT virtual circuit currently exists.	
LAT$_ITM_SERVICE_ GROUPS (BOTH)	A bit mask of LAT group codes that are serviced by the node.	
LAT$_ITM_ PROTOCOL_VERSION (BOTH)	LAT protocol version string.	
LAT$_ITM_DATALINK_ ADDRESS (REMOTE)	LAN address used by the node.	
LAT$_ITM_NODE_ LIMIT	Maximum number of nodes in LAT database. The default is zero, where the maximum is determined by system resources.	
LAT$_ITM_ RETRANSMIT_LIMIT	LAT retransmit limit. Possible values are 4 to 120 retransmissions. The default is 8 retransmissions.	
LAT$_ITM_MAXIMUM_ UNITS (LOCAL)	Maximum LTA unit number.	

(continued on next page)

Table 5–17 (Cont.) LAT$C_ENT_NODE Item Codes

Item Codes	Meaning	
LAT$_ITM_SERVER_ MODE (LOCAL)	Controls whether the node allows the use of the MASTER side of the LAT protocol for outbound connections. Possible values are:	
	LAT$C_DISABLED	Server mode disabled (this is the default).
	LAT$C_ENABLED	Server mode enabled.
LAT$_ITM_SERVICE_ RESPONDER (LOCAL)	Indicates whether the node is to respond to service inquiries originating from a remote system. These inquiries are not necessarily directed at services being offered by the node. See the *OpenVMS System Management Utilities Reference Manual* for more information on this feature. Possible values are:	
	LAT$C_DISABLED	Service responder disabled (this is the default).
	LAT$C_ENABLED	Service responder enabled.
LAT$_ITM_ OUTGOING_SES_ LIMIT (LOCAL)	Maximum number of outgoing LAT sessions. A value of zero, which is the default, indicates that the limit is determined by system resources.	
LAT$_ITM_ INCOMING_SES_ LIMIT (LOCAL)	Maximum number of interactive LAT sessions. A value of zero, which is the default, indicates that the limit is determined by system resources.	
LAT$_ITM_USER_ GROUPS (LOCAL)	Bit mask of LAT group codes to be used when attempting outbound connections using the MASTER side of the LAT protocol.	
LAT$_ITM_CIRCUIT_ TIMER (BOTH)	Circuit timer value in milliseconds. Possible values are 10 to 1000 milliseconds. The default is 80 milliseconds.	
LAT$_ITM_CPU_ RATING (LOCAL)	CPU rating.	
LAT$_ITM_ KEEPALIVE_TIMER (LOCAL)	Keepalive timer value in seconds. Possible values are 10 to 255 seconds. The default is 20 seconds.	
LAT$_ITM_ MULTICAST_TIMER (BOTH)	Multicast timer value in seconds. Possible values are 10 to 180 seconds. The default is 20 seconds.	
LAT$_ITM_ CONNECTIONS (BOTH)	Indicates whether inbound connections (interactive sessions) can be accepted. Possible values are:	
	LAT$C_DISABLED	Inbound connections disabled.
	LAT$C_ENABLED	Inbound connections enabled (this is the default).
LAT$C_ITM_LARGE_ BUFFERS	Indicates in Boolean logic whether the LAT software is using large packet support by default.	
LAT$C_ITM_ ANNOUNCEMENTS	Indicates in Boolean logic whether the LAT software is transmitting LAT service advertisement messages.	

Node service information is presented as a list of node service subblocks, with each subblock containing information about one particular service offered by the node. The subblock item code LAT$_ITM_NODE_SVC_BLOCK has the LAT$V_ STRING bit set to 1, and the string length byte actually contains the length of the entire subblock. Each subblock itself is an itemlist and consists of the item codes listed in Table 5–18.

Table 5–18 Node Service Subblock Item Codes

Item Codes	Meaning	
LAT$_ITM_SERVICE_NAME (BOTH)	Name of a LAT service offered by the node.	
LAT$_ITM_SERVICE_ STATUS (BOTH)	Status of the service. Possible values are:	
	LAT$C_AVAILABLE	Service available
	LAT$C_UNAVAILABLE	Service unavailable
LAT$_ITM_SERVICE_TYPE (LOCAL)	Type of service. Possible values are:	
	LAT$C_ST_GENERAL	Creates a general timesharing service.
	LAT$C_ST_APPLICATION	Creates a special application service that must then be associated with ports dedicated to accepting connections to this service (dedicated ports).
LAT$_ITM_RATING (BOTH)	LAT service rating associated with the service.	
LAT$_ITM_RATING_TYPE (LOCAL)	Type of LAT rating calculation being done by this node. Possible values are:	
	LAT$C_STATIC	Static rating calculation
	LAT$C_DYNAMIC	Dynamic rating calculation
LAT$_ITM_ IDENTIFICATION (BOTH)	Identification string associated with the service.	

Alpha

On Alpha systems, port counters information is presented as a counters subblock. The subblock item code LAT$_ITM_COUNTERS has the LAT$V_STRING bit set to 1, and the string length byte actually contains the length of the entire subblock. The subblock itself is an itemlist and consists of the item codes listed in Table 5–19. ♦

Table 5–19 Node Counters Item Codes for Port Counters Subblocks (Alpha Only)

Item Codes	Meaning
LAT$_ITM_CTPRT_LCL	Indicates number of local accesses to port.
LAT$_ITM_CTPRT_SLCA	Indicates number of solicitations accepted.
LAT$_ITM_CTPRT_SLCR	Indicates number of solicitations rejected.
LAT$_ITM_CTPRT_ISOLA	Indicates number of incoming solicitations accepted.
LAT$_ITM_CTPRT_ISOLR	Indicates number of incoming solicitations rejected.
LAT$_ITM_CTPRT_FRAMERR	Indicates number of framing errors for named port. Returned in port counter subblock.
LAT$_ITM_CTPRT_PARERR	Indicates number of parity errors for named port. Returned in port counter subblock.
LAT$_ITM_CTPRT_OVERRUN	Indicates number of data overruns for named port. Returned in port counter subblock.
LAT$_ITM_PASSWORD_ FAILURES	Indicates password failures.

Node counters information is presented as a counters subblock. The subblock item code LAT$_ITM_COUNTERS has the LAT$V_STRING bit set to 1, and the string length byte actually contains the length of the entire subblock. The subblock itself is an itemlist and consists of the item codes listed in Table 5–20.

Table 5–20 Node Counters Item Codes

Item Codes	Meaning
LAT$_ITM_CTNOD_SSZ (BOTH)	Seconds since zeroed
LAT$_ITM_CTNOD_MSGR (BOTH)	Messages received
LAT$_ITM_CTNOD_MSGT (BOTH)	Messages transmitted
LAT$_ITM_CTNOD_SLTR (BOTH)	Slots received
LAT$_ITM_CTNOD_SLTT (BOTH)	Slots transmitted
LAT$_ITM_CTNOD_BYTR (BOTH)	Bytes received
LAT$_ITM_CTNOD_BYTT (BOTH)	Bytes transmitted
LAT$_ITM_CTNOD_MNA (BOTH)	Multiple node addresses
LAT$_ITM_CTNOD_DUP (BOTH)	Duplicates received
LAT$_ITM_CTNOD_MRT (BOTH)	Messages retransmitted
LAT$_ITM_CTNOD_ILM (BOTH)	Illegal messages received
LAT$_ITM_CTNOD_ILS (BOTH)	Illegal slots received
LAT$_ITM_CTNOD_SLCA (BOTH)	Solicitations accepted
LAT$_ITM_CTNOD_SLCR (BOTH)	Solicitations rejected
LAT$_ITM_CTNOD_TER (LOCAL)	Transmit errors
LAT$_ITM_CTNOD_RES (LOCAL)	Resource errors
LAT$_ITM_CTNOD_NTB (LOCAL)	No transmit buffer
LAT$_ITM_CTNOD_TMO (LOCAL)	Virtual circuit timeout
LAT$_ITM_CTNOD_DOB (LOCAL)	Discarded output bytes
LAT$_ITM_CTNOD_LSTER (LOCAL)	Last transmit error
LAT$_ITM_CTNOD_MCBXMT (LOCAL)	Number of multicast bytes transmitted

(continued on next page)

Table 5–20 (Cont.) Node Counters Item Codes

Item Codes	Meaning
LAT$_ITM_CTNOD_MCBRCV (LOCAL)	Number of multicast bytes received
LAT$_ITM_CTNOD_MCMXMT (LOCAL)	Number of multicast messages transmitted
LAT$_ITM_CTNOD_MCMRCV (LOCAL)	Number of multicast messages received
LAT$_ITM_CTNOD_SOLFAIL (LOCAL)	Number of solicitation failures
LAT$_ITM_CTNOD_ATLOS (LOCAL)	Number of times attention slot data was lost
LAT$_ITM_CTNOD_DATLOS (LOCAL)	Number of times user data was lost
LAT$_ITM_CTNOD_NOREJ (LOCAL)	Number of times a reject slot could not be sent
LAT$_ITM_CTNOD_LOSCT (LOCAL)	Number of times remote node counters were lost
LAT$_ITM_CTNOD_LOSSAM (LOCAL)	Number of service announcement messages lost
LAT$_ITM_CTNOD_NOSAM (LOCAL)	Number of times a service announcement message could not be sent
LAT$_ITM_CTNOD_NOSTS (LOCAL)	Number of times node status was lost
LAT$_ITM_CTNOD_NOXMT (LOCAL)	Number of times no link was available for a transmit
LAT$_ITM_CTNOD_ CTLERR(LOCAL)	Number of controller errors
LAT$_ITM_CTNOD_ CERRCOD(LOCAL)	Lost controller error
LAT$_ITM_CTNOD_ ISOLA(LOCAL)	Number of incoming solicitations accepted
LAT$_ITM_CTNOD_ ISOLR(LOCAL)	Number of incoming solicitations rejected
LAT$_ITM_CTNOD_PROTO (LOCAL)	Protocol error count
‡ LAT$_ITM_CTNOD_XSTR (REMOTE)	Indicates that the node attempted to start up too many LAT sessions for a specific virtual circuit.

‡Alpha specific.

Several protocol errors are also included in a separate subblock. The protocol errors item code is LAT$_ITM_PROTOCOL_ERRORS and has LAT$V_STRING set (the size of the subblock is contained in the first byte following the item code). The item codes and the events they represent are listed in Table 5–21.

Table 5–21 Protocol Error Item Codes

Item Codes	Meaning
LAT$_ITM_CTPRO_IVM (LOCAL)	Invalid message type received
LAT$_ITM_CTPRO_ISM (LOCAL)	Invalid start message received
LAT$_ITM_CTPRO_IVS (LOCAL)	Invalid sequence number received
LAT$_ITM_CTPRO_NIZ (LOCAL)	Zero-node index received
LAT$_ITM_CTPRO_ICI (LOCAL)	Node circuit index out of range
LAT$_ITM_CTPRO_CSI (LOCAL)	Node circuit sequence invalid
LAT$_ITM_CTPRO_NLV (LOCAL)	Node circuit index no longer valid
LAT$_ITM_CTPRO_HALT (LOCAL)	Circuit was forced to halt
LAT$_ITM_CTPRO_MIZ (LOCAL)	Invalid master slot index
LAT$_ITM_CTPRO_SIZ (LOCAL)	Invalid slave slot index
LAT$_ITM_CTPRO_CRED (LOCAL)	Invalid credit field
LAT$_ITM_CTPRO_RCSM (LOCAL)	Repeat creation of slot by master
LAT$_ITM_CTPRO_RDSM (LOCAL)	Repeat disconnection of slot by master
‡ LAT$_ITM_CTPRO_ INVCLASS (LOCAL)	Indicates the number of times a LAT message was received with an invalid service class specified in that message (local node only).
‡ LAT$_ITM_CTPRO_ EXCSTART (LOCAL)	Indicates that a remote node attempted to start up too many LAT sessions. When a virtual circuit is started between two LAT nodes, the maximum number of sessions on that virtual circuit is negotiated. If the master node attempts to create more sessions than the maximum number of sessions on a virtual circuit, then the operating system rejects the excess connections and increments this counter.

‡Alpha specific.

Table 5–22 lists the item codes that are returned for the LAT$C_ENT_SERVICE entity type. As in Table 5–17, item codes noted as LOCAL are returned only if the information being returned is for a locally offered service. Item codes noted as REMOTE are returned only if the information being returned is for a service offered by a remote node. Item codes noted as BOTH are returned for both types of services.

Table 5–22 LAT$C_ENT_SERVICE Item Codes

Item Codes	Meaning	
LAT$_ITM_SERVICE_NAME (BOTH, SUMMARY)	Service name.	
LAT$_ITM_SERVICE_STATUS (BOTH, SUMMARY)	Status of the specified service. Possible values are:	
	LAT$C_AVAILABLE	Service available
	LAT$C_UNAVAILABLE	Service unavailable
LAT$_ITM_SERVICE_TYPE (LOCAL,SUMMARY)	Type of service. Possible values are:	
	LAT$C_ST_GENERAL	General timesharing service
	LAT$C_ST_APPLICATION	Special application service associated with ports dedicated to accepting connections to this service
LAT$_ITM_IDENTIFICATION (BOTH, SUMMARY)	Service identification string, as advertised by the highest rated node that currently offers the service.	

Service node information is presented as a list of service node subblocks, with each subblock containing information about one particular node that offers the service. The subblock item code LAT$_ITM_SVC_NODE_BLOCK has the LAT$V_STRING bit set to 1, and the string length byte actually contains the length of the entire subblock. Each subblock itself is an itemlist and consists of the item codes listed in Table 5–23.

Table 5–23 Service Node Subblock Item Codes

Item Codes	Meaning	
LAT$_ITM_NODE_NAME (BOTH)	Name of a LAT node that offers the selected service.	
LAT$_ITM_STATE (LOCAL)	Current state of the LAT protocol on the local node. Possible values are:	
	LAT$C_ON	New connections are allowed, and the LAT protocol is running.
	LAT$C_OFF	New connections are not allowed, and any current connections are abnormally terminated. The LAT protocol is not running.
	LAT$C_SHUT	No new connections are allowed. Currently active connections are still maintained. The LAT protocol remains running only until the last active session is disconnected, at which time the node is placed in the OFF state.

(continued on next page)

Table 5–23 (Cont.) Service Node Subblock Item Codes

Item Codes	Meaning	
LAT$_ITM_NODE_STATUS (REMOTE)	Current status of the remote node. This item code is present only if a LAT virtual circuit does not currently exist to the remote node. Possible values are:	
	LAT$C_REACHABLE	Remote node is reachable.
	LAT$C_UNREACHABLE	Remote node is unreachable.
	LAT$C_UNKNOWN	Remote node status is unknown.
LAT$_ITM_CONNECTED_ COUNT (REMOTE)	Number of LAT sessions from the local node to this remote node. This item code replaces the LAT$_ITM_ NODE_STATUS item code for remote nodes to which a LAT virtual circuit currently exists.	
LAT$_ITM_RATING (BOTH)	LAT service rating associated with the service.	
LAT$_ITM_RATING_TYPE (LOCAL)	Type of LAT rating calculation being done by this node. Possible values are LAT$C_STATIC and LAT$C_ DYNAMIC.	
LAT$_ITM_ IDENTIFICATION (BOTH)	Identification string associated with the service.	

Service counters information is presented as a counters subblock. The subblock item code LAT$_ITM_COUNTERS has the LAT$V_STRING bit set, and the string length byte actually contains the length of the entire subblock. Each subblock itself is an itemlist and consists of the item codes listed in Table 5–24.

Table 5–24 Service Counters Subblock Item Codes

Item Codes	Meaning
LAT$_ITM_CTSRV_SSZ (BOTH)	Seconds since zeroed.
LAT$_ITM_CTSRV_MCNA (BOTH)	Outgoing connections attempted (the number of times the local node has attempted to connect to the service offered on a remote node).
LAT$_ITM_CTSRV_MCNC (BOTH)	Outgoing connections completed (the number of times the local node successfully connected to the service offered on a remote node).
LAT$_ITM_CTSRV_SCNA (BOTH)	Incoming connections accepted (the number of times the local node has accepted a connection request from a remote node to the locally offered service).
LAT$_ITM_CTSRV_SCNR (BOTH)	Incoming connections rejected (the number of times the local node rejected a connection request from a remote node to the locally offered service).
LAT$_ITM_DED_PORT_ BLOCK (LOCAL)	If the selected service is an application service offered by the local node, a list of one or more port subblocks is included in the itemlist. These subblocks describe the dedicated port or ports associated with this application service, with each subblock describing one particular port. The subblock item code LAT$_ITM_DED_PORT_BLOCK has the LAT$V_STRING bit set, and the string length byte actually contains the length of the entire subblock. Each subblock itself is an itemlist and currently consists only of the following item code: LAT$_ITM_PORT_NAME (LOCAL) — Name of the dedicated port
‡ LAT$_ITM_PASSWORD_ FAILURE	Indicates password failures.

‡Alpha specific.

Table 5–25 lists the item codes that are returned for the LAT$C_ENT_LINK entity type.

Table 5–25 LAT$C_ENT_LINK Item Codes

Item Codes	Meaning
LAT$_ITM_LINK_NAME (SUMMARY)	Link name (such as LAT$LINK).
LAT$_ITM_STATE (SUMMARY)	State of the link. Possible values are: LAT$C_ON — New connections are allowed, and the LAT protocol is running. LAT$C_OFF — New connections are not allowed, and any current connections are abnormally terminated. The LAT protocol is not running.

(continued on next page)

Table 5–25 (Cont.) LAT$C_ENT_LINK Item Codes

Item Codes	Meaning	
	LAT$C_SHUT	No new connections are allowed. Currently active connections are still maintained. The LAT protocol remains running only until the last active session is disconnected, at which time the node is placed in the OFF state.
LAT$_ITM_DEVICE_NAME (SUMMARY)	The name of the LAN device used for the link.	
LAT$_ITM_DATALINK_ ADDRESS	The LAN device's current physical address for the link.	
LAT$_ITM_DECNET_ ADDRESS	Indicates whether the link attempts to use the default DECnet LAN address when starting the data link controller (enabling the LAT protocol). Possible values are:	
	LAT$C_DISABLED	DECnet LAN address use disabled.
	LAT$C_ENABLED	DECnet LAN address use enabled (this is the default).

Link counters information is presented as a counters subblock. The subblock item code LAT$_ITM_COUNTERS has the LAT$V_STRING bit set, and the string length byte actually contains the length of the entire subblock. Because the link counters are independent of the protocol type, they include not only LAT messages and events, but also all other protocol messages and events (that is, DECnet) associated with the same LAN device. The counters are actually maintained by the LAN device driver and are identified within the subblock by the nonprotocol-specific item codes listed in Table 5–26.

Table 5–26 Link Counters Item Codes

Item Codes	Meaning
NMA$C_CTLIN_ZER	Seconds since zeroed
NMA$C_CTLIN_DBR	Messages received
NMA$C_CTLIN_DBS	Messages transmitted
NMA$C_CTLIN_MBL	Multicast messages received
NMA$C_CTLIN_MBS	Multicast messages transmitted
NMA$C_CTLIN_BRC	Bytes received
NMA$C_CTLIN_BSN	Bytes transmitted
NMA$C_CTLIN_MBY	Multicast bytes received
NMA$C_CTLIN_MSN	Multicast bytes transmitted
NMA$C_CTLIN_RFL	Receive errors
NMA$C_CTLIN_SFL	Transmit errors
NMA$C_CTLIN_OVR	Data overrun

(continued on next page)

Table 5–26 (Cont.) Link Counters Item Codes

Item Codes	Meaning
NMA$C_CTLIN_UBU	User buffer unavailable
NMA$C_CTLIN_SBU	System buffer unavailable
NMA$C_CTLIN_LBE	Local buffer errors
NMA$C_CTLIN_BS1	Messages sent, single collisions
NMA$C_CTLIN_BSM	Messages sent, multiple collisions
NMA$C_CTLIN_BID	Messages sent, initially deferred
NMA$C_CTLIN_CDC	Transmit collision detection check failure

Table 5–27 lists additional link counter item codes of the LINK entity.

Table 5–27 Link Counters Item Codes

Item Codes	Meaning
LAT$_ITM_CTLAT_RMSG	Count of LAT messages received through link
LAT$_ITM_CTLAT_RBYT	Count of bytes for LAT received through link
LAT$_ITM_CTLAT_XMSG	Count of LAT messages transmitted through link
LAT$_ITM_CTLAT_XBYT	Count of bytes for LAT transmitted through link
LAT$_ITM_CTLAT_MUL_ RMSG	Count of LAT multicast messages received through link
LAT$_ITM_CTLAT_MUL_ RBYT	Count of multicast bytes for LAT received through link
LAT$_ITM_CTLAT_MUL_ XMSG	Count of LAT multicast messages transmitted through link
LAT$_ITM_CTLAT_MUL_ XBYT	Count of multicast bytes for LAT transmitted through link
LAT$_ITM_LAT_DEV_CTR_ BLOCK	This block contains the LAT specific counters for the specified link. Counters returned in this block are the ones defined above (with CTLAT in their name). These counters are LAT specific for the link (device). They do not include counts from other protocols using the same adapter.

The counter item codes listed in Table 5–27 are used by LATCP in the display generated by the $SHOW LINK /COUNTER command. The display looks similiar to the following:

```
Link Name:    LAT$LINK
Device Name:  _XQA1:
```

```
Seconds Since Zeroed:              65535

Messages Received:               7080630   Messages Sent:               2135394
LAT Messages Received:           1484817   LAT Messages Sent:           2086167
Multicast Msgs Received:         5578139   Multicast Msgs Sent:           10775
LAT Multicast Msgs Received:     5093417   LAT Multicast Msgs Sent:        9142
Bytes Received:                678189475   Bytes Sent:               1312778402
LAT Bytes Received:            107809441   LAT Bytes Sent:           1278118808
Multicast Bytes Received:      602984574   Multicast Bytes Sent:        1696264
LAT Multicast Bytes Received:  565264261   LAT Multicast Bytes Sent:    1448342
System Buffer Unavailable:       1638401   User Buffer Unavailable:           1
Unrecognized Destination:          65537   Data Overrun:                      1
Receive Errors:                        7   Transmit Errors:                   1

Receive Errors (bitmask = 001) -          Transmit Errors (bitmask = 001) -
   Block Check Error:               Yes       Excessive Collisions:        Yes
   Framing Error:                   No        Carrier Check Failure:       No
   Frame Too Long:                  No        Short Circuit:               No
   Frame Status Error:              No        Open Circuit:                No
   Frame Length Error:              No        Frame Too Long:              No
                                              Remote Failure To Defer:     No
                                              Transmit Underrun:           No
                                              Transmit Failure:            No

CSMACD Specific Counters
------------------------

Transmit CDC Failure:                 1

Messages Transmitted -
   Single Collision:               5208
   Multiple Collisions:            4732
   Initially Deferred:                0
```

Table 5–28 lists the item codes that are returned for the LAT$C_ENT_PORT entity type.

Table 5–28 LAT$C_ENT_PORT Item Codes

Item Codes	Meaning	
LAT$_ITM_PORT_NAME SUMMARY	Name of the port (such as _LTA15:).	
LAT$_ITM_PORT_TYPE SUMMARY	Type of port. Possible values are:	
	LAT$C_PT_FORWARD	Forward port used for outgoing LAT connections or for management functions
	LAT$C_PT_INTERACTIVE	Interactive port created as the result of an incoming LAT connection request
	LAT$C_PT_APPLICATION	Application port for solicited connections
	LAT$C_PT_DEDICATED	Dedicated port associated with a local service

(continued on next page)

Table 5–28 (Cont.) LAT$C_ENT_PORT Item Codes

Item Codes	Meaning	
LAT$_ITM_QUEUED	Controls whether the solicited connection requests queued or nonqueued access. Possible values are:	
	LAT$C_DISABLED	Queued access disabled.
	LAT$C_ENABLED	Queued access enabled (this is the default).
LAT$_ITM_SERVICE_ CLASS	Indicates the class driver with which the device is communicating. This item code can be used only with an entity status of LAT$C_ENTS_NEW. Therefore, the service class must be specified when the device is created. An attempt to change the service class of an existing device returns SS$_BADPARAM. Possible values are:	
	LAT$C_SERVCLASS_ INTERACTIVE	Service class 1, TTDRIVER (this is the default).
	LAT$C_SERVCLASS_ TESTSERVICE	Service class 2, TEST SERVICE.
	LAT$C_SERVCLASS_ XTRANSPORT	Service class 3, X Protocol.
	LAT$C_SERVCLASS_FONT	Service class 4, X fonts.
LAT$_ITM_DISPLAY_ NUMBER	Display number value for the device. This field has meaning for services classes 3 and 4 (X) only. It returns a value zero for all other service classes.	
LAT$_ITM_DISCONNECT_ REASON	Reason (if any) for the last disconnect on the port. If it is not a 0–19 LAT rejection code, it is a LAT message code. The 0–19 LAT rejection code meanings are listed in Table 5–32.	
LAT$_ITM_CONNECTED_ SERVICE_NAME[1]	Name of service to which this port is connected. For forward and application ports, this is the name of the remote service to which the port is connected (if any). For interactive and dedicated ports, this is the name of the local service that accepted the remote-initiated connection.	
LAT$_ITM_CONNECTED_ NODE_NAME[1]	Name of remote node to which this port is connected.	
LAT$_ITM_CONNECTED_ PORT_NAME[1]	Name of remote port to which this port is connected.	
LAT$_ITM_CONNECTED_ LINK_NAME[1]	Name of the link on whick the LAT connection exists.	
LAT$_ITM_TARGET_ SERVICE_NAME[2]	Target service name for connection of forward or application ports. For dedicated ports, this item code specifies the local service with which the port is associated.	
LAT$_ITM_TARGET_NODE_ NAME[2]	Target node name for connection of forward or application ports.	
LAT$_ITM_TARGET_PORT_ NAME[2]	Target port name for connection of forward or application ports.	

[1]Returned only when the LTA port has an active LAT connection.

[2]Shows information about how the port is set up. May be returned even if there is no current LAT connection.

(continued on next page)

Table 5–28 (Cont.) LAT$C_ENT_PORT Item Codes

Item Codes	Meaning	
‡ LAT$_ITM_NODE_ QUEUE_POSITION	Indicates current node queue position for connect request. Returned during SENSEMODE of port entity.	
‡ LAT$_ITM_SERVICE_ QUEUE_POSITION	Indicates current service queue position for connect request. Returned during SENSEMODE of port entity.	
LAT$_ITM_PORT_STATE	Current port state. Possible values are:	
	LAT$C_PT_STATE_ INACTIVE	Port is inactive.
	LAT$C_PT_STATE_ CONNECTING	Port connection in progress but not complete.
	LAT$C_PT_STATE_ACTIVE	Port has active LAT connection.
	LAT$C_PT_STATE_ DISCONNECTING	PORT LAT connection in process of terminating.

‡Alpha specific.

Alpha

On Alpha systems, the item codes for queue entries are listed in Table 5–29. ♦

Table 5–29 LAT SENSMODE Queue Entries (Alpha Only)

Item Code	Meaning
LAT$_ITM_QUEUED_ENTRY_ ID (SUMMARY)	Indicates by string the queue entry ID name.
LAT$_ITM_NODE_QUEUE_ POSITION (SUMMARY)	Indicates the current position of entry in node wide queue.
LAT$_ITM_SERVICE_QUEUE_ POSITION (SUMMARY)	Indicates the current position of entry in service wide queue.
LAT$_ITM_NODE_NAME (SUMMARY)	Indicates where the remote node name queue entry came from.
LAT$_ITM_SERVICE_NAME (SUMMARY)	Indicates the target service name to which the queue entry is queued (if specified).
LAT$_ITM_PORT_NAME (SUMMARY)	Indicates the target port name to which the entry is queued (if specified).
LAT$_ITM_LINK_NAME	Returns the link name on which the queued request is active.
LAT$_ITM_DATALINK_ ADDRESS	Returns the remote node that issued request's data link address.

5.4.4.5 Programming Application Ports

An application port is used to connect to a remote device (typically a printer) on a terminal server or to a dedicated port on another LAT service node. The LAT port driver can only connect to a remote device if the device is currently not in use. Table 5–30 lists the conditions that can occur when an application program issues an IO$M_LT_CONNECT request for a connection to a remote device. After a request is queued on the terminal server (or dedicated port on another LAT service node), the QIO request is not completed until the connection is established, rejected, or times out.

Table 5–30 IO$M_LT_CONNECT Request Status

Event	IOSB Status	Explanation
Connection established	SS$_NORMAL	The connection is successful, and the port is ready for use.
Connection timeout	SS$_TIMEOUT	The connection did not complete because communication was never established with the remote end. IOSB+2 contains LAT$_CONTIMEOUT.
Connection rejected	SS$_ABORT. IOSB+2 contains LAT rejection code or LAT facility message code.	The connection cannot be made. The LAT port driver updates the I/O status block. The LAT rejection codes (0–19) are listed in Table 5–32.
Connection request	SS$_ILLIOFUNC	The QIO request is not to an application, dedicated, or forward port. The LAT port driver rejects the request immediately.
Connection already established on port	SS$_DEVACTIVE	The QIO request is for a port already in use. The LAT port driver rejects the request immediately.
Incorrectly configured LAT port	SS$_DEVREQERR	The LAT port is incorrectly configured. This may mean that the port type was neither forward nor application nor dedicated, because a forward port had no service name mapped or because an application port had no node name mapped.
Insufficient resources	SS$_INSFMEM	The QIO request failed because the LAT port driver could not get system memory to complete the connection.

Before the application port can be used, it must be mapped to a remote node name, and either the port name or the service name of the remote terminal server port. (These names must be defined locally on the terminal server.) The application port is mapped with the IO$M_LT_SETMODE modifier, specifying the following items in the P1 itemlist parameter:

- LAT$_ITM_TARGET_NODE_NAME—The node name. The node name is the name of the terminal server where the application device is located.
- LAT$_ITM_TARGET_PORT_NAME—The port name.
- LAT$_ITM_TARGET_SERVICE_NAME—The service name.

The queued status of the connection can also be mapped to the port by specifying the LAT$_ITM_QUEUED item in the P1 itemlist parameter. Valid values for this item are:

- LAT$C_ENABLED—Port has queued status. This is the default.
- LAT$C_DISABLED—The port does not have queued status.

5.4.4.6 Programming Application Services and Dedicated Ports

Rather than the normal timesharing service offered by the operating system, application programs can make use of LAT application services that allow terminal server users (or users on sytems with outgoing connections) to connect to a specialized application. To do this, the system manager must create LAT ports that are dedicated to a particular application service. (Alternatively, this LAT port creation can be done from a program using the QIOs discussed in previous sections, providing OPER privilege.) When the remote user makes the connection to the application service, the connection is directly to the application program that controls a LAT port (LTA device) associated with the service. In this case the prompt, Username:, is not received. Digital recommends that you follow these steps to create an application service:

1. Define the dedicated ports in LAT$SYSTARTUP.COM and execute the command procedure in SYSTARTUP_VMS.COM. (Refer to the *OpenVMS System Management Utilities Reference Manual* and the *OpenVMS System Manager's Manual* for additional information.)
2. Run the application program. Within the application program, allocate dedicated ports with the same name as those defined in LAT$SYSTARTUP.COM. Use the Assign I/O Channel ($ASSIGN) system service to assign service channels to the ports.
3. Post a read request to the dedicated ports. When the terminal user connects to the service and presses the Return key, the application program can perform I/O to the dedicated port.
4. To break the connection, use the Deassign I/O Channel ($DASSGN) system service to deassign the channel and the Deallocate Device ($DALLOC) system service to deallocate the device. The application program must reallocate the port and reassign the channel in preparation for the next connection.

An example of the application service concept is a program that provides the time of day. For this example, the system manager includes the following lines in LAT$SYSTARTUP.COM (or enters them manually in the LATCP program):

```
CREATE SERVICE TIME/ID="At the tone, the time will be"/APPLICATION
CREATE PORT LTA99:/DEDICATED
SET PORT LTA99:/SERVICE=TIME
```

An application program then assigns a channel to device LTA99. When a terminal server user types CONNECT TIME, the user is connected to this application program, and the program prints out the time of day. The program then deassigns the channel, which disconnects the server user.

A system manager may associate more than one LAT port with the same service. In that case, the application program that offers the service should assign channels to all of the LTA devices created for that service.

5.4.4.7 Programming Forward Ports

An outbound LAT connection to a remote service node can be made using a forward port. The LAT port driver can connect to a remote service node only if outgoing connections are enabled on the local node. Outgoing connections can be enabled with LATCP or with a LAT SETMODE QIO to the local node. In addition, user group codes on the local node must match the service group codes of the service to which they are being connected. LATCP can list the services to which the local node can connect. (See the *OpenVMS System Management Utilities Reference Manual* for additional information.) Before the forward port can be used to make an outbound LAT connection, it must be mapped to a service

and optionally, a node and port. The forward port is mapped with the IO$M_LT_SETMODE modifier, specifying the following items in the P1 item list parameter:

- LAT$_ITM_TARGET_SERVICE_NAME—The service name. The service name is the name of the service to which to connect.
- LAT$_ITM_TARGET_NODE_NAME—The node name. The node name is the name of a specific service node offering the service.
- LAT$_ITM_TARGET_PORT_NAME—The port name. The port name is the name of a specific port on the target node. The LAT$_ITM_TARGET_NODE_NAME item must be supplied when supplying this item.
- LAT$_ITM_SERVICE_PASSWORD—The password. The password is required for access to a password-protected service.

A LAT SETMODE QIO on a forward port does not require OPER privilege if the port name is not specified in the P4 parameter. In other words, the LAT SETMODE QIO must be to the port corresponding to the CHAN parameter (the forward port attained by assigning a channel to _LTA0:). Note that SS$_NOPRIV is returned if you attempt to change the port type by specifying the LAT$_ITM_PORT_TYPE item code in the P1 itemlist parameter. If the P4 parameter is specified, the LAT port driver also returns SS$_NOPRIV.

Table 5–30 lists the conditions that can occur when an application program issues an IO$M_LT_CONNECT request for a connection to a remote service node. The QIO request is completed when a session is established with the service node. Once the connection completes, data can be read and written to the port with the QIO read and write functions.

5.4.4.8 Queue Change Notification (Alpha Only)

Alpha

On Alpha systems, the IO$M_LT_QUE_CHG_NOTIF function modifier for $QIO allows a process to enable an attention asynchronous system trap (AST), which is used with the LAT $QIO connect request. The IO$M_LT_QUE_CHG_NOTIF function is available only for APPLICATION and FORWARD LAT devices.

If a $QIO connect request has been issued to a remote node and that request has been queued, this attention AST will be set each time the queue position changes. This AST can be used as long as the $QIO connect request is queued. Like a Ctrl/y AST, it is set only once; it must be reenabled after each completion.

If the LAT $QIO connect succeeds or if a LAT connection exists for the intended service, the AST completes with the SS$_DEVACTIVE status code.

If the LAT device does not have the queued characteristic, issuing the IO$M_LT_QUE_CHG_NOTIF function results in the return of SS$_DEVREQERR status code.

The implementation of IO$M_LT_QUE_CHG_NOTIF is shown in the following C example:

```
status = sys$qiow  (
                0,                  /* efn                */
                ltchannel,          /* channel            */
                IO$_TTY_PORT|IO$M_LT_QUE_CHG_NOTIF,
                                    /* function           */
                q_iosb,             /* iosb               */
                0,                  /* astadr             */
                0,                  /* astprm             */
                queue_pos_change,   /* P1 = ast routine   */
                0, 0, 0, 0, 0);     /* P2 through P6 not used */
```

When a queue position change occurs, the AST routine is called with a 32-bit value. If this value is 0, then the LAT connect $QIO is about to complete, if it has not already. If the value is not 0, the lower word of 16 bits indicates the service queue position, and the upper word of 16 bits indicates the node queue position.♦

5.4.4.9 Hangup Notification

To allow notification by the terminal driver of abnormal termination during I/O operations, enable a Ctrl/Y AST on the channel. This ensures that the terminal driver notifies application programs of an abnormal connection termination. Note that the operating system does not return an AST parameter to the Ctlr/y AST routine.

When an application with a pending read or write request has an abnormal LAT connection completion, the terminal driver returns a SS$_HANGUP status in the first word of the IOSB. The reason for the abnormal LAT connection completion can be attained with a LAT SENSEMODE QIO request to the port. Search the resulting P1 itemlist for the value corresponding to the LAT$_ITM_ DISCONNECT_REASON item code. The value is either a LAT reject code or a LAT facility message. The LAT$V_SENSE_FULL bit must be set in the P3 parameter in order to receive this information.

If IOSB indicates an abnormal completion (SS$_ABORT, see Table 5–30) on a IO$M_LT_CONNECT modifier QIO, the LAT port driver returns the reason for the abnormal completion in IOSB+2. The reason can also be attained with the LAT SENSEMODE QIO function.

5.4.5 Sense Mode and Sense Characteristics

The sense mode and sense characteristics functions sense the characteristics of the terminal and return them to the caller in the I/O status block. The following function codes are provided:

- IO$_SENSEMODE
- IO$_SENSECHAR

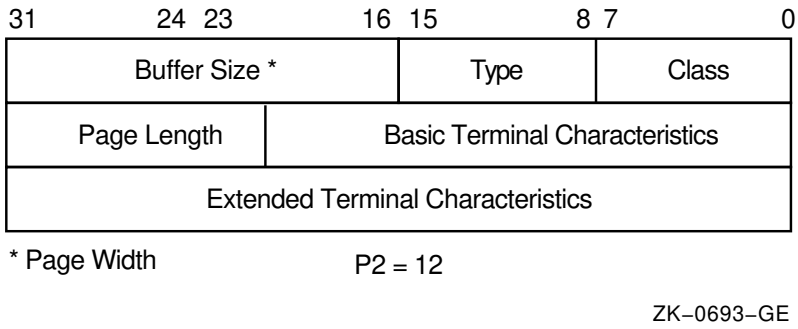
IO$_SENSEMODE returns the temporary characteristics of the terminal (the characteristics associated with the current process), and IO$_SENSECHAR returns the permanent characteristics of the terminal. IO$_SENSEMODE is a logical I/O function and requires no privilege. IO$_SENSECHAR is a physical I/O function and requires the privilege necessary to perform physical I/O.

These function codes take the following device- or function-dependent arguments:

- P1—Address of a characteristics buffer
- P2—Length of characteristics buffer (default length is 8 bytes)

  For remote terminals, specify a P2 value of 8 or 12 only.

Figure 5–11 Sense Mode Characteristics Buffer



The P1 argument points to a variable-length block, as shown in Figure 5–11.

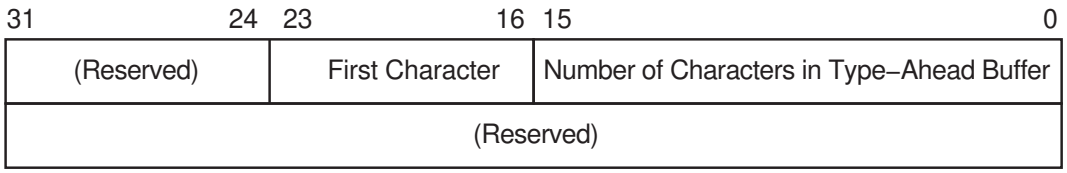
In the buffer, the device class is DC$_TERM, which is defined by the $DCDEF macro. The terminal type is defined by the $TTDEF macro, such as TT$_LA36. The maximum entry for buffer size (page width) is 255. Table 5–5 lists the values for terminal characteristics. Table 5–6 lists the extended terminal characteristics. Characteristics values are defined by the $TTDEF macro.

The sense mode and sense characteristics functions can take the type-ahead count, read modem, and broadcast function modifiers described in the next few sections.

5.4.5.1 Type-ahead Count Function Modifier

The type-ahead count function modifier returns the count of characters presently in the type-ahead buffer and a copy of the first character in the buffer. In this case, the P1 argument points to a characteristics buffer returned by IO$M_ TYPEAHDCNT. Figure 5–12 shows the format of this buffer.

Figure 5–12 Sense Mode Characteristics Buffer (type-ahead)



ZK–0694–GE

5.4.5.2 Read Modem Function Modifier

The read modem function modifier allows access to controller-dependent information. The following combinations of function code and modifier are provided:

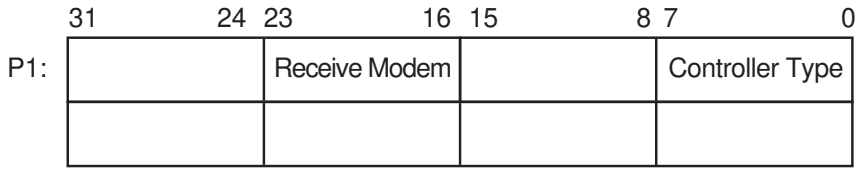
- IO$_SENSEMODE!IO$M_RD_MODEM
- IO$_SENSECHAR!IO$M_RD_MODEM

These function code modifier pairs take the following device- or function-dependent argument:

- P1—The address of a quadword block

Figure 5–13 shows the format of this block.

Figure 5–13 Sense Mode P1 Block



ZK–0695–GE

The receive modem field returns the value of the current input modem signals. Any or all of the following signals can be returned:

- TT$M_DS_DSR—Data set ready (DSR)
- TT$M_DS_RING—Calling indicator (RING)
- TT$M_DS_CARRIER—Data channel received line signal detector (CARRIER)
- TT$M_DS_CTS—Ready for sending (CTS)
- TT$M_DS_SECREC—Received backward channel data (Sec RxD)

The $TTDEF macro defines the symbols for the receive modem field.

The controller type field returns the type of terminal controller in use by the currently active terminal line. The $DCDEF macro defines the symbols for the following types of controllers:

- DT$_DZ11—DZ11 and DZV11
- DT$_DZ32—DZ32
- DT$_DMF32—DMF32

- DT$_DMB32—DMB32
- DT$_DMZ32—DMZ32
- DT$_DHV—DHV11
- DT$_DHU—DHU11
- DT$_LAT—LAT server

Note 1

For LAT devices, the receive modem field of the IO$M_RD_MODEM function modifier does not return any valid modem signal data.

Note 2

The IO$M_RD_MODEM function modifier is not supported for remote terminals. The status SS$_DEVREQERR is returned in the I/O status block.

5.4.5.3 Broadcast Function Modifier

The broadcast function modifier returns those bits that have been set by the set mode function modifier IO$M_BRDCST (see Table 5–12 in Section 5.4.3.6). The following combination of function code and modifier is provided:

- IO$_SENSEMODE!IO$M_BRDCST

This function code modifier pair takes the following device- or function-dependent arguments:

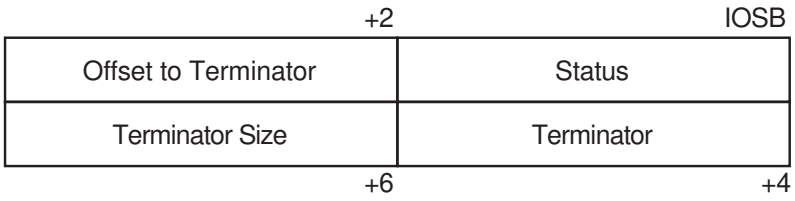
P1—A buffer that contains the bits that specify the requester IDs to be broadcast. (If the bit is set in the first longword, that particular command is turned off for broadcast.)
P2—The length of the P1 buffer.

5.5 I/O Status Block

The I/O status block (IOSB) formats for the read, write, set mode, set characteristics, sense mode, sense characteristics, and LAT port driver I/O functions are shown in Figures 5–14, 5–16, 5–17, and 5–18. Figure 5–15 shows the IOSB format for the itemlist read function. Appendix A lists the status returns for these functions. (The OpenVMS system messages documentation provides explanations and suggested user actions for these returns.)

In Figure 5–14, the offset to terminator at IOSB+2 is the count of characters before the terminator character (see Section 5.4.1.2). The terminator character is in the buffer at the offset specified in IOSB+2. When the buffer is full, the offset at IOSB+2 is equal to the requested buffer size. At the same time, IOSB+4 is equal to 0. In the case of multiple character escape sequences that act as terminators, the terminator at IOSB+4 is the first character (ESC) of the escape sequence. IOSB+6 contains the size of the terminator string, usually 1. However, in an escape sequence, IOSB+6 contains the size of the validated escape sequence (see Section 5.2.1.4). The sum of IOSB+2 and IOSB+6 is the number of characters in the buffer.

Figure 5–14 IOSB Contents—Read Function



ZK–0696–GE

In Figure 5–15 the terminator position word contains a number, the character of which is determined by the mode of operation. For itemlist read operations that do not specify TRM$K_EM_RDVERIFY, this word contains the number of characters from the end of the buffer to the cursor location at the time the terminator character was received. If TRM$K_EM_RDVERIFY is specified, the terminator position word contains the offset into the buffer from the nonverified character.

Figure 5–15 IOSB Contents—Itemlist Read Function

Offset to Terminator		Status	
Cursor Position from EOL	Terminator Length	(Reserved)	Terminator Character

IOSB Contents: Itemlist Read Function

ZK–1306–GE

The byte at IOSB+5 passes the status information listed in Table 5–31 on TRM$K_EM_RDVERIFY operations in which TRM$M_TM_ARROWS or TRM$M_TM_TOGGLE is set in TRM$_MODIFIERS.

Table 5–31 Byte IOSB+5 Status Information

Bit	Interpretation
7 (sign bit)	0 to indicate rest of bits valid. This applies to insert /overstrike and arrow key read verify functionality only.
6–2	Always 0 if bit 7 is equal to 0. Not used; reserved for future use.
1 TRM$V_ST_OTHERWAY	Set to indicate that read is terminated in left-justify insert mode or right-justify overstrike mode.
0 TRM$V_ST_FIELD_FULL	Read terminated on an auto-tab field full condition. IOSB+7 contains an index to the cursor.

In Figure 5–16, the remote terminal driver does not return the number of lines output or the cursor position.

Figure 5–16 IOSB Contents—Write Function

Byte Count	Status
0	0

IOSB Contents: Write Function

ZK–1307–GE

In Figure 5–17, the TTdriver attempts to return the correct data in IOSB after a SETMODE or SETCHAR. To be sure the returned data is correct, the user should follow the SETMODE or SETCHAR with a SENSEMODE or SENSECHAR.

Figure 5–17 IOSB Contents—Set Mode, Set Characteristics, Sense Mode, and Sense Characteristics Functions

Receive Speed *	Transmit Speed	Status	
0	Parity Flags	LF Fill Count	CR Fill Count

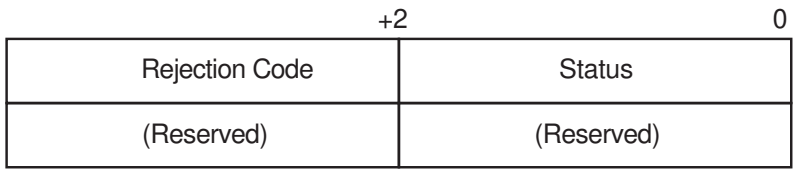
* Only specified if different than transmit speed.

ZK–0698–GE

When an application program makes an I/O request for a connection to a remote device on a terminal server, the LAT port driver places status information about the request into the first word of the I/O status block, as shown in Figure 5–18. Table 5–30 lists the possible status returns.

If the server rejects the request, the LAT port driver returns a numeric LAT rejection code in the second word of the I/O status block. Table 5–32 lists the LAT rejection codes.

Figure 5–18 IOSB Contents—LAT Port Driver Function



ZK–6135–GE

Table 5–32 LAT Rejection Codes

Value	Reason
0	Reason is unknown.
1	User requested disconnect.
2	System shutdown in progress.
3	Invalid slot received.
4	Invalid service class received.
5	Insufficient resources to satisfy request.
6	Service in use.
7	No such service.
8	Service is disabled.
9	Service is not offered on the requested port.
10	Port name is unknown.
11	Invalid password.
12	Entry is not in queue.
13	Immediate access rejected (server queue full).
14	Access denied (group code mismatch).
15	Corrupted solicit request.
16	COMMAND_TYPE code is illegal/not supported.
17	Start slot cannot be sent.
18	Queue entry deleted by local node.
19	Inconsistent or illegal request parameters.

5.6 Terminal Driver Programming Examples

The VAX C program LAT.C shown in Example 5–1 initiates and maintains an outbound LAT session from the local node. It demonstrates the following LAT $QIO functions:

- Cloning the LAT template device (LTA0:)
- IO$M_LT_SETMODE
- IO$M_LT_CONNECT (on forward port)
- IO$M_LT_SENSEMODE

Example 5–1 LAT.C Terminal Driver Programming Example

```
#module LAT_FORWARD_CONNECT "X1.0-001"

/*
**++
**
**  MODULE DESCRIPTION:
**
**      In initiating and maintaining an outbound LAT session from the local
**      node, this program demonstrates the following LAT $QIO functions:
**
**          o Cloning the LAT template device (LTA0:)
**          o IO$M_LT_SETMODE
**          o IO$M_LT_CONNECT   (on forward port)
**          o IO$M_LT_SENSEMODE
**
**--
*/

/*
**
**  INCLUDE FILES
**
*/
#include <descrip> /* VMS Descriptor Definitions      */
#include <iodef>   /* I/O Function Codes Definitions  */
#include <latdef>  /* LAT Definitions                 */
#include <ssdef>   /* System Service Return Status    */
                                    /* Code Definitions                */
#include <ttdef>   /* Terminal Characteristics        */
#include <tt2def>  /* Terminal Extended               */
                                    /* Characteristics                 */
/*
**
**  MACRO DEFINITIONS
**
*/

/*
** Service name which the session will be to.
*/

#define SERVICE_NAME        "LAT_SERVICE"
#define SERVICE_NAME_LENGTH 11
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
/*
** For the sake of clarity, the sizes of the buffers used for reading from
** and writing to the LTA and TT devices are set to the values below.  In
** order to gain maximum throughput from this program, the following SYSGEN
** parameters can be set:
**
**      o TTY_ALTYPAHD - 1500
**      o TTY_TYPAHDSZ - 80
**
** To get the best performance from this program without touching these
** SYSGEN parameters on your system, modify the program to set the size of
** the buffers to the following:
**
**      o LTA_BUFFER_SIZE = MIN(TTY_ALTYPAHD, 1500)
**      o TT_BUFFER_SIZE  = MIN(TTY_TYPAHDSZ, 132)
*/

#define LTA_MAXBUF              1500
#define TT_MAXBUF               80

/*
** Size of the LAT SENSEmode itemlist.
*/

#define MAX_SENSE_ITEMLIST_SIZE 1500

/*
** Character user can press to terminate the LAT connection (CTRL+\).
*/

#define CONNECTION_TERMINATOR   0x1C

/*
**
**  FUNCTION PROTOTYPES
**
*/

unsigned long   SetDeviceChars(void);
void            ConnectAST(void);
void            LTAreadChannelAST(void);
void            TTreadChannelAST(void);
void            LTAhangupHandler(void);
void            EndSession(void);
void            ExitHandler(void);

/*
**
**  GLOBAL DATA
**
*/

char            *LTAbuffer, /* LTA device I/O buffer               */
                *TTbuffer,  /* TT device  I/O buffer               */

                /*
                ** Text for LAT reject codes.  Note that some LAT
                ** implementations will return a 0 reject code to
                ** indicate a normal disconnect.
                */
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
                *LATrejectTable[] = {
                        "Unknown",
                        "User requested disconnect",
                        "System shutdown in progress",
                        "Invalid slot received",
                        "Invalid service class received",
                        "Insufficient resources at server",
                        "Port or service in use",
                        "No such service",
                        "Service is disabled",
                        "Service is not offeredon the requested port",
                        "Port name is unknown",
                        "Invalid service password",
                        "Remote entry is not in queue",
                        "Immediate access rejected",
                        "Access denied",
                        "Corrupted request",
                        "Requested function is not supported",
                        "Session cannot be started",
                        "Queue entry deleted by server",
                        "Illegal request parameters"
                };

unsigned short  LTAchannel,         /* LTA device I/O channel              */
                TTchannel,          /* TT device I/O channel               */
                LTA_QIOiosb[4],     /* IOSB for LTA device functions       */
                TT_QIOiosb[4];      /* IOSB for TT device functions        */

unsigned long   ReadTerminatorMask[2] = { 0, 0 },
                                    /* $QIO read terminator mask           */
                SavedTTdeviceChar[3],
                                    /* Saved TT device characteristics     */
                DeviceCharBuffSize = sizeof(SavedTTdeviceChar);
                                    /* Size of device characteristics buffer*/
                ExitConditionValue, /* Exit condition value of program     */
                LATrejectTableSize =/* Number of elements in LAT reject tbl */
                    sizeof(LATrejectTable) / sizeof(LATrejectTable[0]);

/*
** Itemlist for setting LAT port with the target service name.
*/

struct {
        unsigned short  item_code;
        char            item_byte_count;
        char            item_value[ SERVICE_NAME_LENGTH ];
} PortSetmodeItemlist = {
        LAT$_ITM_TARGET_SERVICE_NAME, SERVICE_NAME_LENGTH, SERVICE_NAME
};

/*
** Exit handler block.
*/

struct {
        unsigned long   flink;
        void            (*exit_handler)();
        unsigned long   arg_count;
        unsigned long   *exit_status;
} ExitHandlerBlock = { 0, ExitHandler, 1, &ExitConditionValue };
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
/*
** Devices which channels are assigned to.
*/

$DESCRIPTOR(LTAtemplateDSC, "LTA0:");
$DESCRIPTOR(TTchannelDSC, "SYS$COMMAND");


main()
{
        /*
        ** Local Variables:
        */

        unsigned long        status,
                        portSetmodeItemlistSize = sizeof(PortSetmodeItemlist);

        /*
        ** BEGIN:
        **
        ** Declare an exit handler.
        */

        if (!((status = sys$dclexh(&ExitHandlerBlock)) & 1))
                lib$signal(status);

        /*
        ** Assign a channel to LTA0: to get a forward LAT port and assign a
        ** channel to the terminal.
        */

        if (!((status = sys$assign(&LTAtemplateDSC, &LTAchannel, 0, 0)) & 1))
                lib$signal(status);
        if (!((status = sys$assign(&TTchannelDSC, &TTchannel, 0, 0)) & 1))
                lib$signal(status);

        /*
        ** Allocate memory for the channel data buffers.
        */

        LTAbuffer = malloc(LTA_MAXBUF);
        TTbuffer = malloc(TT_MAXBUF);

        /*
        ** Set device characteristics for the two channels.
        */

        if (!((status = SetDeviceChars()) & 1))
                lib$signal(status);

        /*
        ** Do SETmode $QIO to set the port entity with the target service name
        ** specified in the item list.
        */
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
        if (!((status = sys$qiow(
                            0,
                            LTAchannel,
                            IO$_TTY_PORT|IO$M_LT_SETMODE,
                            &LTA_QIOiosb, 0, 0,
                            &PortSetmodeItemlist,
                            portSetmodeItemlistSize,
                            LAT$C_ENT_PORT|(LAT$C_ENTS_OLD << 0x10),
                            0, 0, 0)) & 1))
                lib$signal(status);
        if (!(LTA_QIOiosb[0] & 1))
                lib$signal(LTA_QIOiosb[0]);

        /*
        ** Enable a CTRL+Y AST on the LAT channel.
        */

        if (!((status = sys$qiow(
                            0,
                            LTAchannel,
                            IO$_SETMODE|IO$M_CTRLYAST,
                            &LTA_QIOiosb, 0, 0,
                            LTAhangupHandler,
                            0, 0, 0, 0, 0)) & 1))
                lib$signal(status);
        if (!(LTA_QIOiosb[0] & 1))
                lib$signal(LTA_QIOiosb[0]);

        /*
        ** Post the first read (with AST) on the LTA device to ensure that the
        ** first burst of data from the target service is not lost.  It is very
        ** important that the first read is queued before doing the connect
        ** $QIO to ensure no data lossage.
        */

        if (!((status = sys$qio(
                            0,
                            LTAchannel,
                            IO$_READVBLK|IO$M_NOECHO,
                            &LTA_QIOiosb,
                            LTAreadChannelAST, 0,
                            LTAbuffer,
                            1, 0, &ReadTerminatorMask, 0, 0)) & 1))
                lib$signal(status);

        /*
        ** Do the LAT connect $QIO and hibernate until program exit.  The
        ** ConnectAST will execute when the connection completes and post the
        ** initial read on the TT channel.
        */

        if (!((status = sys$qio(
                            0,
                            LTAchannel,
                            IO$_TTY_PORT|IO$M_LT_CONNECT,
                            &LTA_QIOiosb,
                            ConnectAST, 0, 0, 0, 0, 0, 0, 0)) & 1))
                lib$signal(status);
        sys$hiber();
}       /* END - main() */
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
/*
**++
**
**  FUNCTIONAL DESCRIPTION:
**
**      This routine sets device characteristics of the LTA and TT devices.
**      The HOSTSYNC, NOBRDCST, EIGHTBIT and PASTHRU characteristics are set
** on the LTA device.  The ESCAPE and TTSYNC characteristics are cleared.
**
**      The TTSYNC, HOSTSYNC, EIGHTBIT, and PASTHRU characteristics are set
**      on the TT device.  The ESCAPE characteristic is cleared.  The TT
**      characterisitcs are also saved for restoration at program exit.
**
**--
*/

unsigned long   SetDeviceChars(void)
{
        /*
        ** Local Variables:
        */

        unsigned long   status,
                        deviceChar[3];

        /*
        ** BEGIN:
        **
        ** Mask and set the characteristics of the LTA device.  Sense the
        ** current characteristics, and mask in and set the new ones.
        */

        if (!((status = sys$qiow(
                            0,
                            LTAchannel,
                            IO$_SENSEMODE,
                            &LTA_QIOiosb, 0, 0,
                            &deviceChar,
                            DeviceCharBuffSize, 0, 0, 0, 0)) & 1))
                lib$signal(status);
        if (!(LTA_QIOiosb[0] & 1))
                lib$signal(LTA_QIOiosb[0]);

        deviceChar[1] =
            (deviceChar[1] | (TT$M_HOSTSYNC | TT$M_NOBRDCST | TT$M_EIGHTBIT))
     & ~TT$M_ESCAPE & ~TT$M_TTSYNC;
        deviceChar[2] |= TT2$M_PASTHRU;

        if (!((status = sys$qiow(
                            0,
                            LTAchannel,
                            IO$_SETMODE,
                            &LTA_QIOiosb, 0, 0,
                            &deviceChar,
                            DeviceCharBuffSize, 0, 0, 0, 0)) & 1))
                lib$signal(status);
        if (!(LTA_QIOiosb[0] & 1))
                lib$signal(LTA_QIOiosb[0]);
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
        /*
        ** Repeat the procedure for TT device characteristics.  However, save
        ** the current characteristics for restoration at program exit.
        */

        if (!((status = sys$qiow(
                            0,
                            TTchannel,
                            IO$_SENSEMODE,
                            &TT_QIOiosb, 0, 0,
                            &SavedTTdeviceChar,
                            DeviceCharBuffSize, 0, 0, 0, 0)) & 1))
                lib$signal(status);
        if (!(TT_QIOiosb[0] & 1))
                lib$signal(TT_QIOiosb[0]);

        deviceChar[0] = SavedTTdeviceChar[0];
        deviceChar[1] = (SavedTTdeviceChar[1] |
            (TT$M_TTSYNC | TT$M_HOSTSYNC | TT$M_EIGHTBIT)) & ~TT$M_ESCAPE;
        deviceChar[2] = SavedTTdeviceChar[2] | TT2$M_PASTHRU;

        if (!((status = sys$qiow(
                            0,
                            TTchannel,
                            IO$_SETMODE,
                            &TT_QIOiosb, 0, 0,
                            &deviceChar,
                            DeviceCharBuffSize, 0, 0, 0, 0)) & 1))
                lib$signal(status);
        if (!(TT_QIOiosb[0] & 1))
                lib$signal(TT_QIOiosb[0]);

        return(status);
}       /* END - SetDeviceChars */


/*
**++
**
**  FUNCTIONAL DESCRIPTION:
**
**      This routine is an AST which executes when the connect $QIO completes.
**      First the IOSB is checked.  If the connection timed out or was aborted,
**      simply end the session.  Any other abnormal status causes the program
**      to exit.
**
**      Otherwise the connection completed successfully and a read on the TT
**      channel is posted.
**
**--
*/

void    ConnectAST()
{
        /*
        ** Local Variables:
        */

        unsigned long   status;
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
        /*
        ** BEGIN:
        **
        ** If the status in the IOSB indicates that the connection timed out
        ** or aborted, call the session end routine.  Any other abnormal
        ** status causes program exit.
        */

        if ((LTA_QIOiosb[0] == SS$_TIMEOUT) || (LTA_QIOiosb[0] == SS$_ABORT))
                EndSession();

        if (!(LTA_QIOiosb[0] & 1))
                sys$exit(LTA_QIOiosb[0]);

        /*
        ** The connection completed successfully!  Post a read (with AST) on
        ** the TT device and return.
        */

        if (!((status = sys$qio(
                              0,
                              TTchannel,
                              IO$_READVBLK|IO$M_NOECHO,
                              &TT_QIOiosb,
                              TTreadChannelAST, 0,
                              TTbuffer,
                              1, 0, &ReadTerminatorMask, 0, 0)) & 1))
                lib$signal(status);

        return;

}       /* END - ConnectAST */


/*
**++
**
**  FUNCTIONAL DESCRIPTION:
**
**      This routine is an AST which executes when the first character read on
**      the LTA channel completes.  It does a "flush" read of the channel to
**      drain any data out of the ALTYPAHD buffer and writes the data to the
**      TT channel.  It then posts another read on the channel.
**
**--
*/

void    LTAreadChannelAST(void)
{
        /*
        ** Local Variables:
        */

        unsigned long   status;

        /*
        ** BEGIN:
        **
        ** If the status in the IOSB indicates channel hangup, simply end the
        ** session.  Signal any other abnormal status.
        */
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
        if (LTA_QIOiosb[0] == SS$_HANGUP)
                EndSession();
        if (!(LTA_QIOiosb[0] & 1))
                lib$signal(LTA_QIOiosb[0]);

        /*
        ** Do a "flush" read of the LTA device.  This is done by doing a timed
        ** read with a 0 timeout.  There may or may not be any data to drain.
        ** This method is more efficient than using single character reads.
        */

        if (!((status = sys$qiow(
                                0,
                                LTAchannel,
                                IO$_READVBLK|IO$M_TIMED|IO$M_NOECHO,
                                &LTA_QIOiosb, 0, 0,
                                LTAbuffer+1,
                                LTA_MAXBUF-1, 0,
                                &ReadTerminatorMask, 0, 0)) & 1))
                lib$signal(status);
        if (!(LTA_QIOiosb[0] & 1) && (LTA_QIOiosb[0] != SS$_TIMEOUT))
                lib$signal(LTA_QIOiosb[0]);

        /*
        ** The second word of the IOSB contains the number of characters
        ** read.  Write the characters plus 1 for the initial read to the
        ** TT device.
        */

        if (!((status = sys$qiow(
                                0,
                                TTchannel,
                                IO$_WRITEVBLK,
                                &TT_QIOiosb, 0, 0,
                                LTAbuffer,
                                LTA_QIOiosb[1]+1, 0, 0, 0, 0)) & 1))
                lib$signal(status);
        if (!(TT_QIOiosb[0] & 1))
                lib$signal(TT_QIOiosb[0]);

        /*
        ** Post another read on the LTA device.
        */

        if (!((status = sys$qio(
                                0,
                                LTAchannel,
                                IO$_READVBLK|IO$M_NOECHO,
                                &LTA_QIOiosb,
                                LTAreadChannelAST, 0,
                                LTAbuffer,
                                1, 0, &ReadTerminatorMask, 0, 0)) & 1))
                lib$signal(status);

        return;
}       /* END - LTAreadChannelAST */
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
/*
**++
**
**  FUNCTIONAL DESCRIPTION:
**
**      This routine is an AST which executes when the first character read on
**      the TT channel completes.  It does a "flush" read of the channel to
**      drain any data out of the TYPAHD buffer and writes the data to the
**      LTA channel.  It then posts another read on the channel.
**
**--
*/


void    TTreadChannelAST(void)
{
        /*
        ** Local Variables:
        */

        unsigned long   status;

        /*
        ** BEGIN:
        **
        ** If the user pressed the connection terminator character, do a LAT
        ** disconnect $QIO and exit.
        */

        if (*TTbuffer == CONNECTION_TERMINATOR)
        {
                if (!((status = sys$qiow(
                                        0,
                                        LTAchannel,
                                        IO$_TTY_PORT|IO$M_LT_DISCON,
                                        &LTA_QIOiosb, 0, 0, 0, 0, 0, 0, 0)) & 1))
                        lib$signal(status);
                if (!(LTA_QIOiosb[0] & 1))
                        lib$signal(LTA_QIOiosb[0]);
                return;
        }

        /*
        ** Do a "flush" read of the TT device.  This is done by doing a timed
        ** read with a 0 timeout.  There may or may not be any data to drain.
        */

        if (!((status = sys$qiow(
                            0,
                            TTchannel,
                            IO$_READVBLK|IO$M_TIMED|IO$M_NOECHO,
                            &TT_QIOiosb, 0, 0,
                            TTbuffer+1,
                            TT_MAXBUF-1, 0,
                            &ReadTerminatorMask, 0, 0)) & 1))
                lib$signal(status);
        if (!(TT_QIOiosb[0] & 1) && (TT_QIOiosb[0] != SS$_TIMEOUT))
                lib$signal(TT_QIOiosb[0]);
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
        /*
        ** The second word of the IOSB contains the number of characters
        ** read.  Write the characters plus 1 for the initial read to the
        ** TT device.
        */

        if (!((status = sys$qiow(
                            0,
                            LTAchannel,
                            IO$_WRITEVBLK,
                            &LTA_QIOiosb, 0, 0,
                            TTbuffer,
                            TT_QIOiosb[1]+1, 0, 0, 0, 0)) & 1))
                lib$signal(status);

        /*
        ** If the status in the IOSB indicates channel hangup, simply end
        ** the session.  Signal any other abnormal status.
        */

        if (LTA_QIOiosb[0] == SS$_HANGUP)
                EndSession();
        if (!(LTA_QIOiosb[0] & 1))
                lib$signal(LTA_QIOiosb[0]);

        /*
        ** Post another read on the LTA device.
        */

        if (!((status = sys$qio(
                            0,
                            TTchannel,
                            IO$_READVBLK|IO$M_NOECHO,
                            &TT_QIOiosb,
                            TTreadChannelAST, 0,
                            TTbuffer,
                            1, 0, &ReadTerminatorMask, 0, 0)) & 1))
                lib$signal(status);

        return;
}       /* END - TTreadChannelAST */


/*
**++
**
**  FUNCTIONAL DESCRIPTION:
**
**      This routine is the CTRL+Y AST for the LTA channel.  It executes when
**      a hangup on the LTA channel is recognized (connection timed out or
**      aborted).  It will call the session end routine if it hasn't already
**      been called by ConnectAST.
**
**      NOTE:   CTRL+Y ASTs for application ports will NOT execute when the
**              connection is disconnected.
**
**--
*/
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
void    LTAhangupHandler(void)
{
        /*
        ** BEGIN:
        **
        ** Call the session end routine and return.
        */

        EndSession();
        return;

}       /* END - LTAhanghupHandler */

/*
**++
**
**  FUNCTIONAL DESCRIPTION:
**
**      This routine is executed at session end.  It will do a $QIO SENSEmode
**      and search the resulting itemlist to find the reason for the LAT
**      disconnect.  The reason for the disconnect is displayed on the
**      terminal and the image exits.
**
**--
*/

void    EndSession(void)
{

        /*
        ** Local Variables:
        */

        struct ITEM_ENTRY   *itemlistEntry;
        unsigned long       status;
        char                *senseItemlist = malloc(MAX_SENSE_ITEMLIST_SIZE),
                            *itemlistEntryPointer;

        /*
        ** BEGIN:
        **
        ** Do the SENSEmode on the port.
        */

        if (!((status = sys$qiow(
                            0,
                            LTAchannel,
                            IO$_TTY_PORT|IO$M_LT_SENSEMODE,
                            &LTA_QIOiosb, 0, 0,
                            senseItemlist,
                            MAX_SENSE_ITEMLIST_SIZE,
                            LAT$C_ENT_PORT|(LAT$M_SENSE_FULL << 0x10),
                            0, 0, 0)) & 1))
                lib$signal(status);
        if (!(LTA_QIOiosb[0] & 1))
                lib$signal(LTA_QIOiosb[0]);

        /*
        ** Set up two pointers used to traverse the itemlist.
        */

        itemlistEntry = (struct ITEM_ENTRY *) senseItemlist;
        itemlistEntryPointer = senseItemlist;
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
        /*
        ** Search the itemlist for the LAT$_ITM_DISCONNECT_REASON code to find
        ** out why the connection terminated.
        */

        while (itemlistEntry->LAT$R_ITM_CODE.LAT$W_ITEMCODE !=
                LAT$_ITM_DISCONNECT_REASON)
        {
                /*
                ** If the current itemcode being checked has a string value,
                ** advance the pointer to the next itemcode by skipping
                ** BCNT bytes plus 3 bytes for the BCNT byte itself and the
                ** 2 byte itemcode.
                */

                if (itemlistEntry->
                    LAT$R_ITM_CODE.LAT$R_ITM_BITS.LAT$V_STRING)
                        itemlistEntryPointer +=
                            itemlistEntry->LAT$R_ITEM_VALUE.
                                LAT$R_ITEM_COUNTED_STRING.LAT$B_ITEM_BCNT + 3;

                /*
                ** If the current itemcode being checked has a scalar value,
                ** advance the pointer to the next itemcode by skipping 6
                ** bytes for the itemcode and the 4 byte scalar.
                */

                else
                        itemlistEntryPointer += 6;
                itemlistEntry = (struct ITEM_ENTRY *) itemlistEntryPointer;
        }

        /*
        ** If the disconnect reason is a LAT reject code, print out the
        ** text corresponding to the code and set the exit condition value
        ** to SS$_NORMAL.
        */

        if (itemlistEntry->LAT$R_ITEM_VALUE.LAT$L_ITEM_SCALAR_VALUE <=
            LATrejectTableSize)
        {
                printf("\nSession disconnected.  Reason: %s\n\n\n",
                    LATrejectTable[ itemlistEntry->LAT$R_ITEM_VALUE.
                                    LAT$L_ITEM_SCALAR_VALUE ]);
                ExitConditionValue = SS$_NORMAL;
        }

        /*
        ** The scalar value is a LAT facility message code.  Set the exit
        ** condition value to be the scalar.  Upon image exit, the
        ** corresponding LAT facility message will be displayed.
        */

        else
                ExitConditionValue =
                    itemlistEntry->LAT$R_ITEM_VALUE.LAT$L_ITEM_SCALAR_VALUE;

        sys$exit(ExitConditionValue);

}        /* END - EndSession */
```

(continued on next page)

Example 5–1 (Cont.) LAT.C Terminal Driver Programming Example

```
/*
**++
**
**  FUNCTIONAL DESCRIPTION:
**
**      This is the program exit handler which is executed upon image exit.
**      It will cancel all pending I/O on the two channels and restore the
**      TT channel characteristics.
**
**--
*/


void    ExitHandler(void)
{
        /*
        ** Local Variables:
        */

        unsigned long   status;

        /*
        ** BEGIN:
        **
        ** Cancel I/O on the channels, reset terminal characteristics and
        ** return.
        */

        if (!((status = sys$cancel(LTAchannel)) & 1))
                lib$signal(status);
        if (!((status = sys$cancel(TTchannel)) & 1))
                lib$signal(status);

        if (!((status = sys$qiow(
                            0,
                            TTchannel,
                            IO$_SETMODE,
                            &TT_QIOiosb, 0, 0,
                            &SavedTTdeviceChar,
                            DeviceCharBuffSize, 0, 0, 0, 0)) & 1))
                lib$signal(status);
        if (!(TT_QIOiosb[0] & 1))
                lib$signal(TT_QIOiosb[0]);

        return;

}       /* END - ExitHandler */
```

The VAX MACRO program FULL_DUPLEX_TERMINAL.MAR (Example 5–2) shows several I/O operations using the full-duplex capabilities of the terminal. This program illustrates some important concepts about terminal driver programming: assigning an I/O channel, performing full-duplex I/O operations, enabling Ctrl/C AST requests, and itemlist read operations. The program is designed to run with a terminal set to full-duplex mode. The initialization code queues a read request to the terminal and enables Ctrl/C AST requests. The main loop then prints out a random message every three seconds. When you enter a message on the terminal, the read AST routine prints an acknowledgment message and queues another read request. If you press Ctrl/C, the associated AST routine cancels the I/O operation on the assigned channel and exits to the command interpreter.

The VAX MACRO program READ_VERIFY.MAR (Example 5–3) illustrates the read verify function. The program shows a typical build of itemlists (both the right and left fields), channel assignment, a right- and left-justified read verify operation, and then the read QIO operation.

Example 5–2 FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
        .TITLE  FULL_DUPLEX TERMINAL PROGRAMMING EXAMPLE
        .IDENT  /05/
; ****************************************************************
;
;                       TERMINAL PROGRAM
;
; ****************************************************************
        .SBTTL  DECLARATIONS
        .DISABLE        GLOBAL

;
; Declare the external symbols and MACRO libraries.
;
        .EXTERNAL       LIB$GET_EF
        .LIBRARY        'SYS$LIBRARY:LIB.MLB'
        .LIBRARY        'SYS$LIBRARY:STARLET.MLB'
;
; Define symbols
;
        $IODEF                  ; Define I/O function codes
        $QIODEF                 ; Define QIO definition codes
        $SSDEF                  ; Define the system service status codes
        $TRMDEF                 ; Define itemlist read codes
        $TTDEF                  ; Terminal characteristic definitions

;
; Define macros
;
        .SHOW
        .MACRO  ITEM    LEN=0,CODE,VALUE
        .WORD   LEN
        .WORD   TRM$_'CODE'
        .LONG   VALUE
        .LONG   0
        .ENDM   ITEM
        .NOSHOW

;
; Declare exit handler control block
;
EXIT_HANDLER_BLOCK:
        .LONG   0               ; System uses this for pointer
        .LONG   EXIT_HANDLER    ; Address of exit handler
        .LONG   1               ; Argument count for handler
        .LONG   STATUS          ; Destination of status code
STATUS: .BLKL   1               ; Status code from $EXIT

;
; Allocate terminal descriptor and channel number storage
;
```

(continued on next page)

Example 5–2 (Cont.) FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
TT_DESC:
        .ASCID  /SYS$INPUT/     ; Logical name of terminal
TT_CHAN:
        .BLKW   1               ; TT channel number storage

;
; Define acknowledgment message.  This is done right above input buffer
; so that we can concatenate the two together when the acknowledgment
; message is issued.
;
ACK_MSG:
        .ASCII  <CR><LF> /Following input acknowledged: /
ACK_MSGLEN=.-ACK_MSG            ; Calculate length of message

;
; Allocate input buffer
;
IN_BUFLEN = 20                  ; Set length of buffer
IN_BUF:
        .BLKB   IN_BUFLEN       ; Allocate character buffer
IN_IOSB:
        .BLKQ   1               ; Input I/O status block
;
; Define out-of-band ast character mask
;
CNTRLA_MASK:
        .LONG   0
        .LONG   ^B0010          ; Control A mask


;
; Define old terminal characteristics buffer
;
OLDCHAR_BUF_LEN = 12
OLDCHAR_BUF:
        .BLKB   OLDCHAR_BUF_LEN

;
; Define new terminal characteristics buffer
;
NEWCHAR_BUF_LEN = 12
NEWCHAR_BUF:
        .BLKB   NEWCHAR_BUF_LEN

;
; Define carriage control symbols
;
        CR=^X0D                 ; Carriage return
        LF=^X0A                 ; Line feed

;
; Define output messages
;
; Output messages are accessed by indexing into a table of
; longwords with each message described by a message address and
; message length
;
```

(continued on next page)

Example 5–2 (Cont.) FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
ARRAY:                          ; Table of message addresses and
                                ; lengths
        .LONG   10$             ; First message address
        .LONG   15$             ; First message length
        .LONG   20$
        .LONG   25$
        .LONG   30$
        .LONG   35$
        .LONG   40$
        .LONG   45$
;
; Define messages
;
10$:    .ASCII            <CR><LF>/RED ALERT! RED ALERT!/
15$=.-10$
;
20$:    .ASCII            <CR><LF>/ALL SYSTEMS GO/
25$=.-20$
;
30$:    .ASCII            <CR><LF>/WARNING..INTRUDER ALARM/
35$=.-30$
;
40$:    .ASCII            <CR><LF>/** SYSTEM OVERLOAD **/
45$=.-40$
;
; Static QIO packet for message output using QIO$_G form
;

WRITE_QIO:
        $QIO    EFN=SYNC_EFN, - ; QIO packet
                FUNC=IO$_WRITEVBLK!IO$M_BREAKTHRU!IO$M_REFRESH, -
                IOSB=SYNC_IOSB

;
; Declare the required I/O status blocks.
;
SYNC_IOSB::     .BLKQ   1       ; I/O status block for synchronous terminal processing.

;
; Declare the required event flags.
;
ASYNC_EFN::     .BLKL   1       ; Event flag for asynchronous terminal processing.
SYNC_EFN        ==      WRITE_QIO + 4   ; Event flag for sync terminal processing.
TIMER_EFN::     .BLKL   1       ; Event flag for timer processing.

;
; Timer storage
;

WAITIME:
        .LONG   -10*1000*1000*3,-1     ; 3 second delta time
TIME:
        .BLKQ   1               ; Current storage time used for
                                ; random number
```

(continued on next page)

Example 5–2 (Cont.) FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
        .PAGE
        .SBTTL  START - MAIN ROUTINE
        .ENABLE LOCAL_BLOCK
;++
;
; Functional description:
;
; ******************************************************************
;
;                       Start program
;
; ******************************************************************
;
;       The following code performs initialization functions.
;       It is assumed that the terminal is already in
;       FULL-DUPLEX mode.
;
;       NOTE: When doing QIO_S calls, parameters P1 and P3-P6 should be
;             passed by value, while P2 should be passed by reference.
;
; Input parameters:
;       None
;
; Output parameters:
;       None
;
;--
        .ENTRY  START   ^M < >

;               Get the required event flags.

        PUSHAL  ASYNC_EFN
        CALLS   # 1, G^ LIB$GET_EF      ; Get EFN for async terminal operations.
        BLBC    R0, 10$                 ; Error - branch.
        PUSHAL  SYNC_EFN
        CALLS   # 1, G^ LIB$GET_EF      ; Get EFN for sync terminal operations.
        BLBC    R0, 10$                 ; Error - branch.
        PUSHAL  TIMER_EFN
        CALLS   # 1, G^ LIB$GET_EF      ; Get EFN for timer operations.
        BLBC    R0, 10$                 ; Error - branch.

;               Initialize the terminal characteristics.

        $ASSIGN_S       DEVNAM=TT_DESC,-; Assign terminal channel using
                        CHAN=TT_CHAN    ; logical name and channel number
        BLBC    R0, 10$                 ; Error - branch.
        BSBW    CHANGE_CHARACTERISTICS  ; Change the characteristics of
                                        ; terminal
        BSBW    ENABLE_CTRLCAST         ; Allow Ctrl/C traps
        BSBW    ENABLE_OUTBANDAST       ; Enable Ctrl/A out-of-band AST
        BSBW    ENABLE_READ             ; Queue read
        MOVZWL  TT_CHAN, WRITE_QIO+8    ; Insert channel into
        BRB     LOOP                    ; static QIO packet

10$:
        BRW     ERROR

;
; This loop outputs a message based on a random number and then
; delays for 3 seconds
;
```

(continued on next page)

Example 5–2 (Cont.) FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
LOOP:
        $GETTIM_S      TIMADR=TIME      ; Get random time
        BLBC    R0, 10$                 ; Error - branch.
        EXTZV   #6, #2, TIME, R0        ; Load random bits into switch
        MOVQ    ARRAY[R0], -            ; Load message address
                WRITE_QIO+QIO$_P1       ; and size into QIO
                                        ; packet

;
; Issue QIO write using packet defined in data area
;

        $QIOW_G WRITE_QIO
        BLBC    R0, 10$                 ; QIO error - branch.
        MOVZWL  SYNC_IOSB, R0           ; Get the terminal driver status.
        BLBC    R0, 10$                 ; Terminal driver error - branch.


;
; Delay for 3 seconds before issuing next message
;

        $SETIMR_S       EFN=TIMER_EFN,- ; Timer service
                        DAYTIM=WAITIME  ; will set event flag
                                        ; in 3 seconds
        BLBC    R0, 10$                 ; Error - branch.
        $WAITFR_S       EFN=TIMER_EFN   ; Wait for event flag
        BLBS    R0, LOOP                ; No error if set
        BRB     10$                     ; Error - branch.

        .DISABLE        LOCAL_BLOCK

        .PAGE
        .SBTTL  CHANGE_CHARACTERISTICS - CHANGE CHARACTERISTICS OF TERMINAL
;++
;
; Functional description:
;
;       Routine to change the characteristics of the terminal.
;
; Input parameters:
;       None
;
; Output parameters:
;       R0 - status from $QIO call.
;       R1 - R5 destroyed
;
;--
;

CHANGE_CHARACTERISTICS:
        $QIOW_S EFN=SYNC_EFN, -         ; Get current terminal characteristics
                CHAN=TT_CHAN, -
                FUNC=#IO$_SENSEMODE, -
                IOSB=SYNC_IOSB, -
                P1=OLDCHAR_BUF, -
                P2=#OLDCHAR_BUF_LEN
        BLBC    R0, 10$                 ; Error if clear
        MOVZWL  SYNC_IOSB, R0           ; Get the terminal driver status.
        BLBC    R0, 10$                 ; Error - branch
```

(continued on next page)

Example 5–2 (Cont.) FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
        $DCLEXH_S EXIT_HANDLER_BLOCK    ; Declare exit handler to reset
                                        ; characteristics
        BLBC    R0, 10$                 ; Error - branch.
        MOVC3   #OLDCHAR_BUF_LEN, -     ; Move old characteristics into
                OLDCHAR_BUF, -          ; new characteristics buffer
                NEWCHAR_BUF
        BISL2   #TT$M_NOBRDCST, -       ; Set nobroadcast bit
                NEWCHAR_BUF+4           ; ...
        $QIOW_S EFN=SYNC_EFN, -         ; Set current terminal characteristics
                CHAN=TT_CHAN, -
                FUNC=#IO$_SETMODE, -
                IOSB=SYNC_IOSB, -
                P1=NEWCHAR_BUF, -
                P2=#NEWCHAR_BUF_LEN
        BLBC    R0, 10$                 ; QIO error - branch.
        MOVZWL  SYNC_IOSB, R0           ; Get the terminal driver status.
        BLBC    R0, 10$                 ; Terminal driver error - branch.
        RSB
10$:
        BRW     ERROR

        .PAGE
        .SBTTL  ENABLE_CTRLCAST - ENABLE Ctrl/C AST
;++
;
; Functional description:
;
;       Routine to allow Ctrl/C recognition.
;
; Input parameters:
;       None
;
; Output parameters:
;       None
;
;--
;
ENABLE_CTRLCAST:
        $QIOW_S EFN=SYNC_EFN, -
                CHAN=TT_CHAN, -
                FUNC=#IO$_SETMODE!IO$M_CTRLCAST, -
                IOSB=SYNC_IOSB, -
                P1=CTRLCAST, -          ; AST routine address
                P3=#3                   ; User mode
        BLBC    R0, 10$                 ; Error - branch.
        MOVZWL  SYNC_IOSB, R0           ; Get the terminal driver status.
        BLBC    R0, 10$                 ; Terminal driver error - branch.
        RSB
```

(continued on next page)

Example 5–2 (Cont.) FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
10$:
        BRW     ERROR

        .PAGE
        .SBTTL  ENABLE_OUTBANDAST - ENABLE Ctrl/A AST
;++
;
; Functional description:
;
;       Routine to allow CNTRL/A recognition.
;
; Input parameters:
;       None
;
; Output parameters:
;       None
;
ENABLE_OUTBANDAST:
        $QIOW_S EFN=SYNC_EFN, -
                CHAN=TT_CHAN, -
                FUNC=#IO$_SETMODE!IO$M_OUTBAND, -
                IOSB=SYNC_IOSB, -
                P1=CTRLAAST, -          ; AST routine address
                P2=#CNTRLA_MASK, -      ; Character mask
                P3=#3                   ; User mode
        BLBC    R0, 10$                 ; QIO error - branch.
        MOVZWL  SYNC_IOSB, R0           ; Get the terminal driver status.
        BLBC    R0, 10$                 ; Terminal driver error - branch.
        RSB

10$:
        BRW     ERROR

        .PAGE
        .SBTTL  ENABLE_READ - QUEUE A READ TO THE TERMINAL.
;++
;
; Functional description:
;
;       Routine to queue a read operation to the terminal.
;
; Input parameters:
;       None
;
; Output parameters:
;       None
;
; Define item list for itemlist read
;
ITEM_LST:
        ITEM    0, MODIFIERS, -                 ; Convert lowercase to
                TRM$M_TM_CVTLOW!TRM$M_TM_NOEDIT ; upper and inhibit line
        ITEM    6, TERM,MASK_ADDR               ; editing
                                                ; Set up terminator mask
```

(continued on next page)

Example 5–2 (Cont.) FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
ITEM_LEN        =       . - ITEM_LST
MASK_ADDR:
        .LONG   1@^XD                   ; Terminator mask is
                                        ; <CR>
        .WORD   1@4                     ; and "$"
ENABLE_READ:
        $QIO_S  EFN=ASYNC_EFN, -        ; Must not be QIOW form or read will block
                CHAN=TT_CHAN, -         ; process
                FUNC=#IO$_READVBLK!IO$M_EXTEND, -
                IOSB=IN_IOSB, -
                ASTADR=READAST, -       ; AST routine to execute
                P1=IN_BUF, -            ; on
                P2=#IN_BUFLEN, -
                P5=#ITEM_LST, -         ; Itemlist read address
                P6=#ITEM_LEN            ; Itemlist read size
        BLBC    R0, 10$                 ; QIO error - branch.

; The queued read operation will not affect write operations due
; to the fact that breakthru has been set for the write operations.

        RSB

10$:
        BRW     ERROR

        .PAGE
        .SBTTL  READAST - AST ROUTINE FOR READ COMPLETION
        .ENABLE LOCAL_BLOCK
;++
;
; Functional description:
;
;       AST routine to execute on read completion.
;
; Input parameters:
;       None
;
; Output parameters:
;       None
;
;--
;

10$:
        MOVZWL  IN_IOSB, R0             ; Get the terminal driver status
20$:
        BRW     ERROR                   ; Exit with error status.

        .ENTRY  READAST         ^M < R2, R3, R4, R5 >   ; Procedure entry mask

        BLBC    IN_IOSB, 10$            ; Terminal driver error - branch
        MOVZWL  IN_IOSB+2, R0           ; Get number of characters read into R0
        ADDL2   #ACK_MSGLEN, R0         ; Add size of fixed acknowledge message
        $QIO_S  EFN=ASYNC_EFN, -        ; Issue acknowledge message
                CHAN=TT_CHAN, -         ; Note, ACK must be asynchronous (QIO)
                FUNC=#IO$_WRITEVBLK, -  ; and the terminal driver write status
                P1=ACK_MSG, -           ; is ignored (no IOSB and AST routine).
                P2=R0                   ; Specify IOSB and AST routine if output
                                        ; must be displayed on the terminal.
        BLBC    R0, 20$                 ; QIO error - branch
```

(continued on next page)

Example 5–2 (Cont.) FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
;
; Process read message
;
;        .
;        .
;        .
;(user-provided code to decode command inserted here)
;        .
;        .
;        .

        BSBW     ENABLE_READ            ; Queue next read
        RET                             ; Return to mainline loop

        .DISABLE        LOCAL_BLOCK

        .PAGE
        .SBTTL  CTRLAAST - AST ROUTINE FOR Ctrl/A
        .SBTTL  CTRLCAST - AST ROUTINE FOR Ctrl/C
        .SBTTL  ERROR - EXIT ROUTINE
;++
;
; Functional description:
;
;       AST routine to execute when Ctrl/C or Ctrl/A is entered.
;
; Input parameters:
;       None
;
; Output parameters:
;       None
;
CTRLCAST::
CTRLAAST::
        .WORD   ^M < >                  ; Procedure entry mask
        MOVL    #SS$_NORMAL, R0         ; Put success in R0

ERROR::
        $EXIT_S R0                      ; Exit
        RSB

        .PAGE
        .SBTTL  EXIT_HANDLER - EXIT HANDLER ROUTINE
;++
;
; Functional description:
;
;       Exit handler routine to execute when image exits.  It cancels
;       any outstanding I/O on this channel and resets the terminal
;       characteristics to their original state.
;
; Input parameters:
;       None
;
; Output parameters:
;       None
;
;--
;
```

(continued on next page)

Example 5–2 (Cont.) FULL_DUPLEX_TERMINAL.MAR Terminal Driver Programming Example

```
        .ENTRY  EXIT_HANDLER    ^M< >
        $CANCEL_S       CHAN=TT_CHAN    ; Flush any I/O on queue
        $QIOW_S EFN=SYNC_EFN, -         ; Reset terminal characteristics
                CHAN=TT_CHAN, -
                FUNC=#IO$_SETMODE, -
                IOSB=SYNC_IOSB, -
                P1=OLDCHAR_BUF, -
                P2=#OLDCHAR_BUF_LEN
        BLBC    R0, 10$                 ; QIO error - branch.
        MOVZWL  SYNC_IOSB, R0           ; Get the terminal driver status.

10$:
        RET

        .END     START
```

Example 5–3 READ_VERIFY.MAR Terminal Driver Programming Example

```
        .TITLE READ_VERIFY - Read Verify Coding Example
        .IDENT  'V05-000'

        .SBTTL  DECLARATIONS
        .DISABLE        GLOBAL

;
; Declare the external system routines and MACRO libraries.
;
        .EXTERNAL       LIB$GET_EF
        .EXTERNAL       SCR$ERASE_PAGE

        .LIBRARY        'SYS$LIBRARY:LIB.MLB'
        .LIBRARY        'SYS$LIBRARY:STARLET.MLB'
;
; Include files:
;
        $IODEF
        $TRMDEF
;
; Macros:
;
.MACRO ITEM LEN=0,CODE,VALUE
        .WORD   LEN
        .WORD   TRM$_'CODE'
        .LONG   VALUE
        .LONG   0
.ENDM ITEM

;
; Equated symbols:
;
INBUF_LEN = 20
ESC = ^X1B
```

(continued on next page)

Example 5–3 (Cont.) READ_VERIFY.MAR Terminal Driver Programming Example

```
;
; Own storage:
;
; Build item lists for the read verify QIO
;

;
; Right-justified field
;
R_ITEM_LIST:
        ITEM    CODE    = MODIFIERS, -
                VALUE   = TRM$M_TM_R_JUST       ; Right justify

        ITEM    CODE    = EDITMODE, -
                VALUE   = TRM$K_EM_RDVERIFY     ; Enable read verify

        ITEM    CODE    = PROMPT, -
                VALUE   = R_PROMPT_ADDR, -
                LEN     = R_PROMPT_LEN          ; Set up prompt

        ITEM    CODE    = INISTRNG, -
                VALUE   = R_INISTR_ADDR, -
                LEN     = R_INISTR_LEN          ; Set up initial string

        ITEM    CODE    = INIOFFSET, -
                VALUE   = R_INISTR_LEN

        ITEM    CODE    = PICSTRNG, -
                VALUE   = R_PICSTR_ADDR, -
                LEN     = R_PICSTR_LEN          ; Set up picture string

        ITEM    CODE    = FILLCHR, -
                VALUE   = <^A/* />            ; clear = *, fill = space

R_ITEM_LIST_LEN = .-R_ITEM_LIST

R_PROMPT_ADDR:
        .ASCII  <ESC>  /[12;12H$/
R_PROMPT_LEN = .-R_PROMPT_ADDR

R_INISTR_ADDR:
        .ASCII  /   ,   /
R_INISTR_LEN = .-R_INISTR_ADDR

MASK = TRM$M_CV_NUMERIC!TRM$M_CV_NUMPUNC

R_PICSTR_ADDR:
        .BYTE   MASK
        .BYTE   MASK
        .BYTE   MASK
        .BYTE   0               ; Marker character
        .BYTE   MASK
        .BYTE   MASK
        .BYTE   MASK
R_PICSTR_LEN = .-R_PICSTR_ADDR
;
; Left-justified field
;
L_ITEM_LIST:
        ITEM    CODE    = MODIFIERS, -
                VALUE   = TRM$M_TM_CVTLOW!TRM$M_TM_AUTO_TAB
                                                ; Upcase input and
                                                ; complete on field full
```

(continued on next page)

Example 5–3 (Cont.) READ_VERIFY.MAR Terminal Driver Programming Example

```
        ITEM    CODE    = EDITMODE, -
                VALUE   = TRM$K_EM_RDVERIFY     ; Enable read verify

        ITEM    CODE    = PROMPT, -
                VALUE   = L_PROMPT_ADDR, -
                LEN     = L_PROMPT_LEN          ; Set up prompt

        ITEM    CODE    = INISTRNG, -
                VALUE   = L_INISTR_ADDR, -
                LEN     = L_INISTR_LEN          ; Set up initial string

        ITEM    CODE    = INIOFFSET, -
                VALUE   = 0

        ITEM    CODE    = PICSTRNG, -
                VALUE   = L_PICSTR_ADDR, -
                LEN     = L_PICSTR_LEN          ; Set up picture string

        ITEM    CODE    = FILLCHR, -
                VALUE   = <^A/* />              ; clear = *, fill = space
L_ITEM_LIST_LEN = .-L_ITEM_LIST

L_PROMPT_ADDR:
        .ASCII  <ESC>/[13;12H Enter Date: /
L_PROMPT_LEN = .-L_PROMPT_ADDR

L_INISTR_ADDR:
        .ASCII  /  -   -  /
L_INISTR_LEN = .-L_INISTR_ADDR

MASK1 = TRM$M_CV_NUMERIC
MASK2 = TRM$M_CV_UPPER!TRM$M_CV_LOWER


L_PICSTR_ADDR:
        .BYTE   MASK1
        .BYTE   MASK1
        .BYTE   0               ; Marker character
        .BYTE   MASK2
        .BYTE   MASK2
        .BYTE   MASK2
        .BYTE   0               ; marker character
        .BYTE   MASK1
        .BYTE   MASK1
L_PICSTR_LEN = .-L_PICSTR_ADDR

IN_IOSB:        .BLKL   2
TT_CHAN:        .BLKW   1
INBUF:          .BLKB   INBUF_LEN
SYSINPUT:       .ASCID  /SYS$INPUT/
SYNC_EFN:       .BLKL   1

        .PAGE

        .ENTRY  READ_VERIFY     ^M < >

;
; Get the required event flags.
;

        PUSHAL  SYNC_EFN
        CALLS   # 1, G^ LIB$GET_EF
        BLBC    R0, ERROR                       ; Error - branch
;
; Assign the channel to SYS$INPUT
;
```

(continued on next page)

Example 5–3 (Cont.) READ_VERIFY.MAR Terminal Driver Programming Example

```
        $ASSIGN_S -
                CHAN = TT_CHAN -
                DEVNAM = SYSINPUT               ; SYS$INPUT
        BLBC    R0, ERROR                       ; Branch on error

;
; Clear the screen
;

        CLRQ    -(SP)
        CALLS   #2, G^ SCR$ERASE_PAGE
        BLBC    R0, ERROR

;
; Do the right-justified read operation
;

        PUSHL   #R_ITEM_LIST_LEN
        PUSHAB  R_ITEM_LIST
        CALLS   #2, DO_READ
        BLBC    R0, ERROR


;
; Do the left-justified read operation
;

        PUSHL   #L_ITEM_LIST_LEN
        PUSHAB  L_ITEM_LIST
        CALLS   #2, DO_READ
        BLBC    R0, ERROR

ERROR:
        RET

        .PAGE
;++
;
; DO_READ - do the actual QIO
;
; Inputs:
;
;       4(AP)   the address of the itemlist
;       8(AP)   the length of the itemlist
;
;--

        .ENTRY  DO_READ, ^M<>

        $QIOW_S EFN=SYNC_EFN, -
                CHAN = TT_CHAN, -
                FUNC = #<IO$_READVBLK!IO$M_EXTEND>, -
                IOSB = IN_IOSB, -
                p1 = inbuf, -
                p2 = #inbuf_len, -
                p5 = 4(AP), -
                P6 = 8(AP)
        BLBC    R0, 10$                 ; QIO error - branch
        MOVZWL  IN_IOSB, R0             ; Get the terminal driver status.
        BLBC    R0, 10$                 ; Terminal driver error - branch

;  Handle the input...
```

(continued on next page)

Example 5–3 (Cont.) READ_VERIFY.MAR Terminal Driver Programming Example

```
10$:
        RET
        .END READ_VERIFY
```

6
Pseudoterminal Driver

This chapter describes the use of the pseudoterminal driver (FTDRIVER) and the pseudoterminal software.

A pseudoterminal is a software device that appears as a real terminal to an application communicating with it but that does not require the existence of a physical terminal. A pseudoterminal consists of two components: the pseudoterminal device and a control program. The control program acts like a keyboard; that is, anything written to the control program appears on the pseudoterminal device as if the keystrokes had been typed in at a physical terminal. The control program also acts like a viewport to the pseudoterminal device; that is, the control program reads anything that is written by the system to the pseudoterminal device.

A pseudoterminal allows an application to be set up on the control side of the link to communicate with another application that is on the pseudoterminal side. This arrangement allows development of applications that either simulate users or monitor the communication between a real user (at a physical terminal) and an application. As with other devices, the work of the pseudoterminal is performed by a device driver and is tightly coupled to the operating system.

The pseudoterminal driver software includes a set of control connection routines. Applications can use these routines to perform pseudoterminal operations and functions. Appendix C provides the calling conventions for these routines.

6.1 Pseudoterminal Operations

This section contains information on the following pseudoterminal operations:

- Creating a pseudoterminal
- Canceling a request
- Deleting a pseudoterminal

6.1.1 Creating a Pseudoterminal

To create a pseudoterminal, use the PTD$CREATE routine described in Appendix C. When a pseudoterminal is created, it inherits the current system terminal default attributes unless you specify an alternate set of characteristics. In either case, you cannot use PTD$CREATE to alter the following startup attributes:

- TT$M_CRFILL is cleared. To change this attribute, issue the SET MODE $QIO function.
- TT$M_LFFILL is cleared. To change this attribute, issue the SET MODE $QIO function.
- TT$M_MODEM is cleared. This attribute cannot be changed.
- TT$M_REMOTE is cleared. This attribute cannot be changed.

- TT$M_HOSTSYNC is set. To change this attribute, issue the SET MODE $QIO function.
- TT$M_TTSYNC is set. To change this attribute, issue the SET MODE $QIO function.
- TT2$M_DMA is cleared. To change this attribute, issue the SET MODE $QIO function. Changing it does not alter the behavior of TTDRIVER or the pseudoterminal.
- TT2$M_AUTOBAUD is cleared. To change this attribute, issue the SET MODE $QIO function. Changing it does not alter the behavior of TTDRIVER or the pseudoterminal.
- TT2$M_FALLBACK is cleared. To change this attribute, issue the SET MODE $QIO function.
- TT2$M_HANGUP is cleared. To change this attribute, issue the SET MODE $QIO function.
- TT2$M_DCL_MAILBX is cleared. This attribute cannot be changed.

When you create a pseudoterminal, you can specify a repeating asynchronous system trap (AST) to be delivered when the terminal connection is freed. This AST can be supplied only when the pseudoterminal is created, and it cannot be deleted. A terminal is freed when a process logs out or deassigns the last channel to the device. The AST allows the control program to determine whether or not a user of a pseudoterminal is using it. At this point, the control program can reuse or delete the pseudoterminal by deassigning the control channel.

6.1.2 Canceling a Request

To cancel a queued control connection request, the control program uses the PTD$CANCEL routine. This routine enables the pseudoterminal driver to differentiate between control requests and terminal requests that are being canceled. This routine cannot be used to flush event notification ASTs.

6.1.3 Deleting a Pseudoterminal

To delete the pseudoterminal, the control program uses the PTD$DELETE routine. When a pseudoterminal is deleted, any process that is using the pseudoterminal (except the control process) is disconnected. If you have the TT2$M_DISCONNECT bit set in the default terminal characteristics parameter (TTY_DEFCHAR2) and virtual terminals have been enabled (see Section 5.2.2.3), you get a virtual terminal upon logging in to a pseudoterminal. In this case, the process is not logged out, but the virtual terminal is disconnected from the pseudoterminal.

The PTD$DELETE request causes any pending I/O for the control program to be aborted. It deletes any queued event notification ASTs and returns the I/O buffers to the application. It also causes the pseudoterminal unit control block (UCB) to be deleted once the reference count returns to zero.

Note

If an application exits without calling PTD$DELETE, the pseudoterminal is still deleted.

6.2 Pseudoterminal Driver Features

The terminal portion of a pseudoterminal is similar to a regular terminal. The pseudoterminal driver provides the following features:

- Type-ahead buffer
- Specifiable or default line terminators
- Special operating modes, such as NOECHO and PASTHRU
- Escape sequence detection
- Terminal/mailbox interaction
- Terminal control characters, such as Ctrl/S and Ctrl/Q for starting and stopping output, Ctrl/O for discarding output, and all other special characters that are handled by the standard terminal driver
- Limited full-duplex operation (simultaneously active read and write requests)

For more information on these features, see Section 5.2.

6.3 Pseudoterminal Driver Device Information

The pseudoterminal inherits its device characteristics from the system default parameters, with the following exceptions:

- The device inherits initial device characteristics from the SYSGEN-supplied default values. You can modify the device characteristics during device creation by supplying new characteristics.
- The HOSTSYNC terminal characteristic is always set.
- The device is set to NOMODEM and cannot be set to MODEM.
- The device is set not to time output character transmission. Hardware controllers time output character transmission to determine whether the controller is broken.

You can obtain information on pseudoterminal characteristics by using the Get Device/Volume Information ($GETDVI) system service, as described in Section 5.3, and the *OpenVMS System Services Reference Manual*.

Applications should assign a channel other than the control channel in order to read data from, write data to, read, or alter the pseudoterminal characteristics. An attempt to perform such I/O with the control channel, or any other attempt to queue an illegal or unsafe I/O request, results in an SS$_CHANINTLK error.

6.4 I/O Buffers

When you create a pseudoterminal, you must provide at least one page to be used as an I/O buffer.

Alpha

On Alpha systems, you can allocate one page and divide it into I/O buffers as needed. ♦

VAX

On VAX systems, each page becomes one I/O buffer. You should allocate no more than six I/O buffers for each pseudoterminal.♦

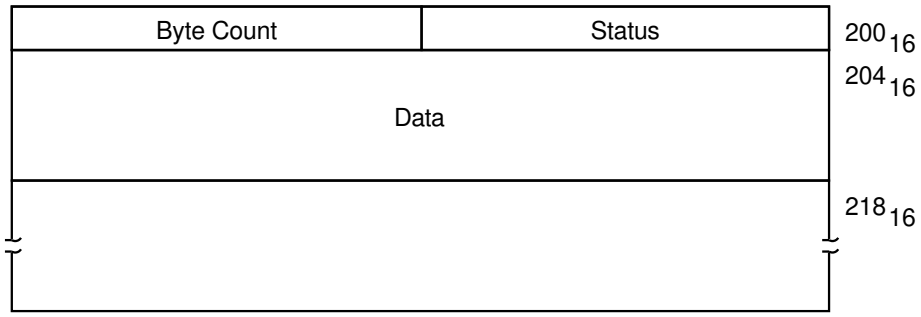
No read or write request should reference more than one I/O buffer at a time. The I/O buffers must be page aligned; therefore, you should create these pages with the $EXPREG system service or the LIB$GET_VM_PAGE routine. The pages are owned by the driver until you delete the pseudoterminal. The application is responsible for managing the pages and cannot use buffers that are owned by another pseudoterminal. The application must decide whether to delete the buffers when they are freed by the driver or to reuse them.

The I/O buffers must be valid pages in virtual address space. Creating or deleting an I/O buffer does not alter the contents of the pages.

The low-order word of the status information longword contains the status of the request. The high-order word of the status information longword contains the actual number of bytes that are read or written.

Assume that an I/O buffer starting at 200 hexadecimal is available for use. If you want to read 20 bytes from the pseudoterminal, the **readbuf** address would be 200, and the **readbuf_len** would be 20. An application can use the rest of this buffer for other purposes, including reading or writing to the pseudoterminal. Figure 6–1 shows how the buffer would look.

Figure 6–1 Buffer Layout



ZK–9656–GE

6.5 Pseudoterminal Functions

This section discusses the following pseudoterminal functions:

- Reading data
- Writing data
- Using write with echo
- Flow control
- Event notification

6.5.1 Reading Data

To read data from the pseudoterminal, the control program uses the PTD$READ routine. The read request completes with a minimum of one character and a maximum of the number of characters requested. The read operation completes when the pseudoterminal has characters to output. If a read request is issued and no data is available, the read request is queued and then completed at a later time.

An application that issues an asynchronous pseudoterminal read can use the $SYNCH system service to find out when the read completed. The **efn** argument for the $SYNCH service must be the same as the **efn** specified in the original PTD$READ call, and the **iosb** for the $SYNCH service must match the **readbuf** of the PTD$READ call.

6.5.2 Writing Data

To write data to the pseudoterminal, the control program uses the PTD$WRITE routine. The write request allows you to specify a buffer to receive any output that the write request generates; you do not need to issue a separate read request to read this data. When you use an echo buffer, the control application can significantly reduce the number of I/O requests required.

An application can issue only one write request at a time. Once the write request completes, the application must check the write buffer status longword to see whether all the data supplied was written. If not, the application must issue additional write requests until all the data has been accepted.

6.5.3 Using Write with Echo

If a read request is pending when a write-with-echo request is issued, the echo data is placed in the echo buffer. If more data is echoed than can fit in the echo buffer, the remaining data is placed in the pending read requests buffer. If no pending read exists, the data is held by the driver until another request that can take the data is issued. Both the read and the write with echo must use completion ASTs to allow the driver to report request completions to the application in the correct order.

If an application is not using the write-with-echo capability, the application should avoid using completion ASTs if possible. Unnecessary use of completion ASTs significantly increases the number of instructions needed to complete a read or write operation.

When using write with echo, both the **wrtbuf** and **echobuf** arguments contain I/O status information. An application must check both of these status longwords if the PTD$WRITE completes successfully. If a write operation wrote no characters, characters might still be in the echo buffer. If no data was echoed, the status in the **echobuf** is SS$_NORMAL with zero bytes transferred.

6.5.4 Flow Control

By default, the driver attempts to notify the control program of data overrun or loss. The pseudoterminal sends an XOFF AST when the type-ahead buffer is getting full. Once the pseudoterminal delivers an XOFF AST, the pseudoterminal also returns a status of SS$_DATAOVERUN with the actual number of characters input. This prevents a single request from flooding the type-ahead buffer. If a control program makes repeated attempts to insert data after receiving the SS$_DATAOVERUN message, it can flood the terminal type-ahead buffer. When

the type-ahead buffer has filled, the pseudoterminal returns the status of SS$_DATALOST.

If the control program is writing to the terminal or terminal driver, it should let the terminal and terminal driver handle flow control. To do this, the application should enable all three input flow control notification ASTs. The control program should write a DC1 to the terminal if an XON AST is delivered. It should write a DC3 to a terminal if an XOFF AST is delivered, and write a BELL character to the terminal if the BELL AST is delivered. These signals allow the terminal to decide what to do with the flow control data. The application should ignore the SS$_DATAOVERUN and SS$_DATALOST return status and continue writing data to the pseudoterminal.

6.5.5 Event Notification

This section describes how the pseudoterminal driver provides notification of important driver events.

6.5.5.1 Input Flow Control

The driver provides three ways to indicate when the class driver wants to stop input and one way to signal when it is safe to resume output.

1. The driver returns a status of SS$_DATAOVERUN and the number of characters input for the control program write.
2. The control program can enable a BELL attention AST to be delivered when the class driver calls the PTD$SET_TERMINAL_NOTIFICATION routine. This AST is delivered if the pseudoterminal does not have the HOSTSYNC attribute set. If only a BELL or only an XOFF AST event is enabled and an XOFF or a BELL AST needs to be delivered, the AST that is available is delivered.
3. The control program can enable an XOFF attention AST to be delivered when the class driver calls the PTD$SET_TERMINAL_NOTIFICATION routine. This AST is delivered if the pseudoterminal has the HOSTSYNC attribute set.
4. The control program can enable an XON attention AST to be delivered when the class driver calls the PTD$SET_TERMINAL_NOTIFICATION routine. This AST is delivered only if the pseudoterminal has the HOSTSYNC attribute set.

6.5.5.2 Output Stop

The Output Stop AST tells the control program that the terminal driver is stopping output. This keeps the control program from having to determine whether an XOFF written to the control side is being treated by the terminal driver as flow control or data.

6.5.5.3 Output Resume

The Output Resume AST tells the control program that the terminal driver wants to resume output. This AST can be delivered at any time, even if output is active or has previously been stopped. The control program should always restart output processing when it receives this AST.

6.5.5.4 Characteristics Changed

The Characteristics Changed AST tells the control program that the terminal driver has called the pseudoterminal CHANGE CHARACTERISTICS routine. This routine is called whenever the terminal driver has changed the device characteristics. The control program should then read the pseudoterminal characteristics to determine what has changed.

6.5.5.5 Output Abort

The Output Abort AST tells the control program that the terminal driver has called the pseudoterminal ABORT OUTPUT routine. This routine is called when the terminal driver wants to flush any outstanding output data. The control program should flush any internally buffered data when this AST is received.

6.5.5.6 Terminal Driver Read Events

Three special event types notify the control program when a terminal read request starts and finishes. By default, the pseudoterminal does not deliver the read notification ASTs associated with these events. The PTD$SET_EVENT_NOTIFICATION routine must be used explicitly to enable or disable their delivery.

- Start Read—Tells the control program that the terminal driver is starting a read request. Some applications require this in order to know when to start inputting a logged session script.
- Middle Read—Tells the control program that the terminal driver has finished writing the prompt string if one was supplied.
- End Read—Tells the control program that the terminal driver has finished a read request.

Once an event notification AST is enabled, it continues to be delivered until it is canceled, or until the device is deleted. This characteristic allows the control program to enable the AST once, thus greatly reducing the risk of missing multiple rapid occurrences of an event. If the driver cannot get sufficient resources to deliver the notification AST, that report is lost. Only one AST per event is allowed, and attempts to specify multiple ASTs result in use of the last one specified.

To enable or disable event notification, the control program uses the PTD$SET_EVENT_NOTIFICATION routine, which is described in Appendix C.

6.6 Pseudoterminal Driver Programming Example

Example 6–1 illustrates how to use the pseudoterminal. (The example is also included in the SYS$EXAMPLES directory.) This section begins with a brief overview of the example. The example itself briefly discusses each module; the pseudocode for that module follows its discussion.

The scenario chosen for this example is a simple terminal session logging utility that uses most of the pseudoterminal capabilities. This example also illustrates how to use the write-with-echo capability, which provides a significant gain in performance.

6.6.1 Design Overview

The design approach writes the log record in a main loop that hibernates when it has no work to do. The loop uses ASTs to read keystrokes from the terminal, write to the pseudoterminal, and write data to the terminal. When a block of characters is written to the terminal, that block is placed into a queue of blocks to be written to the log file, and a wake request is issued. Logging is stopped if you log out of the subprocess, if you enter the stop logging character Ctrl\, or if a severe error occurs during data processing. When any of these events occur, all outstanding log records are written before the program exits.

One major design consideration is how flow control should be handled—either by attempting to enforce flow control, or by letting the terminal and terminal driver handle it. In this example, the terminal and terminal driver handle flow control; the driver sends XON, XOFF, or BELL characters to the terminal as necessary.

One of the six I/O buffers is permanently reserved as the terminal read buffer. This buffer is passed directly to the terminal read $QIO. This eliminates having to move data that is read from the terminal into the read buffer. The other five buffers are placed in a queue and are allocated and deallocated as needed. This pool of buffers reserves the first two longwords to be used as queue headers and traditional IOSBs. The third longword and the I/O status longwords are used by the pseudoterminal driver.

Example 6–1 Sample Pseudocode for Pseudoterminal Driver Program

```
/*
** Main Routine
**
** Function: Intitializes the environment and then hibernates, waiting
** to be awakened.  When awakened, the program checks to see whether it
** is exiting, or whether more log data is available.  If more data is
** available, the data is appended to the current log record and checked
** to see whether a log record should be written.  A log record is written
** either when maxbuf characters are in the log buffer,
** or when it finds a <CR><LF> character pair.  The algorithm
** allows an unlimited number of <NULL> fill characters to occur
** between the <CR> and the <LF>. If the program is
** exiting, it closes the log file,  deletes the pseudoterminal, resets the
** terminal, and exits.
*/
Initialize environments (This includes creating pseudoterminal, the log file
                         and starting up the subprocess.)
```

(continued on next page)

Example 6–1 (Cont.) Sample Pseudocode for Pseudoterminal Driver Program

```
If (Initialization OK) Then
   Do
      while (I/O buffer to log)
         Data size = number of bytes in I/O buff
         For all data in I/O buffer
            If (cr_seen) Then
               If (current char == <LF>) Then
                  write current log buffer
                  reset cr_seen
                  point to start of log buffer
               Else if (current char != <NULL>) Then
                  insert <CR> and current char into log buffer
                  move log buffer ptr over 2 characters
                  reset cr_seen
               Endif
            Else if (current character != <CR>) Then
               insert character into log buffer
               move log buffer ptr over 1 character
            Else
               set cr_seen
            Endif

            If (log buffer ptr >= IOC$GW_MAX-48) Then
               write log buffer
               reset log buffer pointer
               reset cr_seen
            Endif
         Endloop
         Free I/O buffer call free_io_buffers
      Endwhile
      If (not exiting) Then
         Wait for more to do call SYS$HIBER
      Endif
   Until ((exiting) and (no I/O buffers to log))

   close log file
   If ((close failed) and (exit reason is SS$_NORMAL)) Then
      set exit to status to failure reason
   Endif
   If (subprocess still running) Then
      call SYS$FORCEX to run down the subprocess
   Endif
   call PTD$CANCEL to flush all pending pseudoterminal read requests
   call SYS$CANCEL to flush all terminal requests
   call PTD$DELETE to delete the pseudoterminal
   If ((delete failed) and (exit reason is SS$_NORMAL)) Then
      set exit to status to failure reason
   Endif
   reset terminal to startup condition using SYS$QIOW
   If ((terminal reset failed) and (exit reason is SS$_NORMAL)) Then
      exit to status to failure reason
   Endif
Endif
call LIB$SIGNAL and report exit reason
Exit
```

(continued on next page)

Example 6–1 (Cont.) Sample Pseudocode for Pseudoterminal Driver Program

```
/*
**
** Initialization Code
**
** Function: This routine sets the terminal characteristics, creates the
** pseudoterminal, starts up the subprocess, and opens the log file.  If
** any of these steps fail, the program undoes any steps already done and
** returns to the  main routine.
**
*/

read the maximum buffer size from IOC$GW_MAXBUF
Assign a channel to SYS$INPUT
If (assign ok) Then
   Read the terminal characteristics from the terminal
   If (read of terminal characteristics ok) Then
      Open log file with maximum record size of IOC$GW_MAXBUF
      If (open ok) Then
         Create the pseudoterminal with characteristics of terminal
         If (create ok) then
            Place 4 of the buffers on the queue of free I/O buffers
            Copy terminal characteristics and modify them to NOECHO and PASTHRU
            Set the terminal characteristics use modified value
            If (set ok) Then
               Get device name of pseudoterminal use SYS$GETDVI
               If (get ok) Then
                  Create subprocess
                  If (create ok) Then
                     Enable XON, XOFF, BELL, SET_LINE event notification ASTs
                     If (AST setup OK) Then
                        Call PTD$READ to start reading from the pseudoterminal
                                     ASTADR = ft_read_ast
                                     ASTPRM = buffer address
                                     READBUF = I/O buffer + 8
                                     READBUF_LEN = 500
                        If (read ok) Then
                           Call SYS$QIO and read a single character from the
                                        keyboard ASTADR = kbd_read_ast
                           If (read failed) Then
                              Call PTD$CANCEL to flush queued pseudoterminal read
                              Call PTD$DELETE to delete pseudoterminal
                              Reset terminal to original state
                              Close log file and delete it
                           Endif
                        Else
                           Call PTD$DELETE to delete pseudoterminal
                           Reset terminal to original state
                           Close log file and delete it
                        Endif
                     Else
                        Call PTD$DELETE to delete pseudoterminal
                        Reset terminal to original state
                        Close log file and delete it
                     Endif
                  Else
                     Call PTD$DELETE to delete pseudoterminal
                     Reset terminal to original state
                     Close log file and delete it
                  Endif
               Else
                  Call PTD$DELETE to delete pseudoterminal
```

(continued on next page)

Example 6–1 (Cont.) Sample Pseudocode for Pseudoterminal Driver Program

```
                  Reset terminal to original state
                  Close log file and delete it
               Endif
            Else
               Call PTD$DELETE to delete pseudoterminal
               Close log file and delete it
            Endif
         Else
            Close log file and delete it
         Endif
      Endif
   Endif
Endif
/*
** kbd_read_ast
**
** Function: This routine is called every time data is read from the terminal.
** If the program is exiting, then the routine exits without restarting the
** read.   The character read is checked to see if the terminate processing
** character Ctrl\ was entered.  If the terminate processing character was
** entered, the exiting state is set and a SYS$WAKE is issued to start the
** main routine.  Now an attempt is made to obtain an I/O buffer in which
** to store echoed output.  If an I/O buffer is unavailable, a simple
** PTD$WRITE is issued; a PTD$WRITE with echo is issued if a buffer is
** available.  If the write completes successfully, another read is issued
** to the keyboard.
**
*/

If (not exiting) Then
   If (read ok) Then
      Search input data for Ctrl\
      Allocate a read buffer call allocate_io_buffer
      If (got a buffer) Then
         Call PTD$WRITE to write characters to pseudoterminal
                        ASTADR = ft_echo_ast
                        ASTPRM = allocated I/O buffer
                        WRTBUF = read I/O buffer
                        WRTBUF_LEN = number of characters read
                        ECHOBUF = allocated I/O buffer
                        ECHOBUF_LEN = 500
      Else
         Call PTD$WRITE to write characters to pseudoterminal
                        WRTBUF = read I/O buffer
                        WRTBUF_LEN = number of characters read
      Endif
      If (write setup ok)
         If ((write status is ok) or (write status is SS$_DATALOST))
            Issue another single character read to terminal with
                     ASTADR = kbd_read_ast, with data going to read I/O buffer
            If (read setup failed) Then
               Set exit flag
               Set exiting reason to SS$_NORMAL
            Endif
         Else
            Set exit flag
            Set exiting reason to SS$_NORMAL
         Endif
      Else
         Set exit flag
         Set exiting reason to SS$_NORMAL
```

(continued on next page)

Example 6–1 (Cont.) Sample Pseudocode for Pseudoterminal Driver Program

```
      Endif
   Else
      Set exit flag
      Set exiting reason to read failure status
   Endif
   If (exiting) Then
      Wake the mainline call SYS$WAKE
   Endif
Endif
/*
** terminal_output_ast
**
** Function: This routine is called every time an I/O buffer is written
** to the terminal.  If the terminal write request completes successfully,
** it inserts the I/O buffer into the queue of I/O buffers to be logged.
** If the I/O buffer is the only entry on the queue, it issues a SYS$WAKE
** to start the main routine. To prevent spurious wake requests,
** SYS$WAKE is not issued if multiple entries are already on
** the queue.  If a terminal write error occurs, the routine sets the
** exit flag and wakes the main routine.
**
*/
If (terminal write completed ok) Then
   insert I/O buffer onto logging queue
   If (this is only entry on queue) Then
      wake the mainline call SYS$WAKE
   Endif
Else
   set exit flag
   set exiting reason to terminal write error reason
   wake the mainline call SYS$WAKE
Endif
/*
**
** ft_read_ast
**
** Function: This routine is called when a pseudoterminal read request
** completes.  It writes the buffer to the terminal and attempts to start
** another read from the pseudoterminal.  If the program is not exiting,
** this routine writes the buffer to the terminal and does not start another
** pseudoterminal read.
**
*/
If (not exiting)
   If (Pseudoterminal read ok) Then
      write buffer to the terminal ASTADR = terminal_output_ast
      If (write setup ok) Then
         Allocate another read buffer call allocate_io_buffer
         If (got a buffer) Then
            Call PTD$READ to restart reads from the pseudoterminal.
                     ASTADR = ft_read_ast
                     ASTPRM = buffer address
                     READBUF = I/O buffer + 8
                     READBUF_LEN = 500
            If (read setup failed) Then
               Set exit flag
               Set exiting reason to read failure reason
               Wake the mainline call SYS$WAKE
            Endif
         Else
            Set read stopped flag
```

(continued on next page)

Example 6–1 (Cont.) Sample Pseudocode for Pseudoterminal Driver Program

```
        Endif
      Else
         Set exit flag
         Set exiting reason to terminal write failure reason
         Wake the mainline call SYS$WAKE
      Endif
   Else
      Set exit flag
      Set exiting reason to terminal read failure reason
      Wake the mainline call SYS$WAKE
   Endif
Endif
/*
**
** ft_echo_ast
**
** Function: This routine is called if a write to the pseudoterminal used
** an ECHO buffer.  If any data was echoed, the output is written to the
** terminal.  If no data was echoed, the I/O buffer is freed so it can be
** used later. If the program is exiting, this routine exits.
**
*/
If (not exiting) Then
   If (ECHO buffer has data) Then
      Write the buffer to the terminal with ASTADR = terminal_output_ast
      If (error setting up write) Then
         Set exit flag
         Set exiting reason to write failure reason
         Wake mainline call SYS$WAKE
      Endif
   Else
      Free I/O buffer call free_io_buffers
   Endif
Endif
/*
** free_io_buffers
**
** Function: This routine places a free I/O buffer on the queue of available
** I/O buffers. It also restarts any stopped read operations from the
** pseudoterminals.  This routine disables AST delivery while it is running
** in order to synchronize reading and resetting the read stopped flag.
**
*/
If (not exiting) Then
   Disable AST deliver using SYS$SETAST
   If (Pseudoterminal reads not stopped) Then
      Insert I/O buffer on the interlocked queue of free I/O buffers
   Else
      Call PTD$READ to restart reads from the pseudoterminal.
                  ASTADR = ft_read_ast
                  ASTPRM = buffer address
                  READBUF = I/O buffer + 8
                  READBUF_LEN = 500
      If (no error starting read) Then
         Clear read stopped flag
      Else
         Set exit flag
         Set exit reason to read setup reason
      Endif
  Endif
  Enable AST delivery using SYS$SETAST
```

(continued on next page)

Example 6–1 (Cont.) Sample Pseudocode for Pseudoterminal Driver Program

```
Endif
/*
**
** allocate_io_buffer
**
** Function: This routine obtains a free I/O buffer from the queue of
** available I/O buffers.  If the program is exiting, this routine
** returns an SS$_FORCEDEXIT error.
**
*/
If (not exiting) Then
   remove a I/O buffer from the interlocked queue of I/O buffers
   If (queue empty) Then
      exit with reason LIB$_QUEWASEMP
else
   exit with reason SS$_FORCEDEXIT
Endif
/*
** subprocess_exit
**
** Function: This routine is called when the subprocess has completed
** and exited.  This routine checks whether the program is already exiting.
** If not, then the routine indicates that the program is exiting and wakes
** up the main program.
**
*/
If (not exiting) Then
   indicate subprocess no longer running
   set exit status to SS$_NORMAL
   indicate exiting
   call SYS$WAKE to start up main loop
Endif
/*
** xon_ast
**
** Function: This routine is called for the pseudoterminal driver to signal
** that it is ready to accept keyboard input. The routine attempts to send
** an XON character to the terminal by sending XON DC1 using SYS$QIO.
** If the attempt fails, it is not retried.
**
*/
If (not exiting) Then
   call SYS$QIO to send a <DC1> character to the terminal
Endif
/*
** bell_ast
**
** Function: This routine is called when the pseudoterminal driver wants
** to warn the user to stop sending keyboard data.  The routine attempts
** to ring the terminal bell by sending the BELL character to the terminal
** using SYS$QIO.  If the attempt fails, it is not retried.
**
*/
If (not exiting) Then
   call SYS$QIO to send a <BELL> character to the terminal
Endif
```

(continued on next page)

Example 6–1 (Cont.) Sample Pseudocode for Pseudoterminal Driver Program

```
/*
** xoff_ast
**
** Function: This routine is called when the pseudoterminal driver wants to
** signal that it will stop accepting  keyboard input. The routine attempts
** to send an XOFF character to the terminal by sending XOFF DC3 to the
** terminal using SYS$QIO.  If the attempt fails, it is not retried.
**
*/
If (not exiting) Then
   call SYS$QIO to send a <DC3> character to the terminal
Endif
/*
** set_line_ast
**
** Function: This routine is called when the pseudoterminal device
** characteristics  change.  The routine reads the current pseudoterminal
** characteristics, changes the characteristics to set PASTHRU and NOECHO,
** and applies the characteristics to the input terminal.  If the attempt
** to alter the terminal characteristics fails, it is not retried.
**
*/
If (not exiting) Then
   call SYS$QIOW to read the pseudoterminals characteristics
   If (not error) Then
      Set the alter the just read characteristics to have PASTHRU and NOECHO
      attributes
      call SYS$QIO to set the terminal characteristics.
   Endif
Endif
```

7
Shadow-Set Virtual Unit Driver

This chapter provides an overview of Volume Shadowing for OpenVMS, and describes the use of the shadow-set virtual unit driver (SHDRIVER).

7.1 Introduction

Volume Shadowing for OpenVMS ensures that data is available for applications and end users by duplicating data on multiple disks. Because the same data is recorded on multiple disk volumes, if one disk fails, the remaining disk or disks can continue to service I/O requests. This ability to shadow disk volumes is sometimes referred to as **disk mirroring**.

Volume Shadowing supports clusterwide shadowing on all of the following:

- DSA devices, including all DSA disks on local adapters
- DSSI (Digital Storage Systems Interconnect) RF-series disk devices on any VAX and Alpha system
- CI, which is a computer interconnect, Ethernet, and mixed-interconnect interfaces
- MSCP, which is an OpenVMS mass storage control protocol server

You can mount one, two or three compatible disk volumes, including the system disk, to form a **shadow set**. The maximum total number of shadow sets is 130 shadow sets. Each disk in the shadow set is known as a shadow set **member**. Volume Shadowing for OpenVMS logically binds the shadow set devices together and represents them as a single virtual device called a **virtual unit**. This means that multiple members of the shadow set, represented by the virtual unit, appear to applications and users as a single, highly available disk.

Volume Shadowing features include:

- Controller independence. Shadow set members can be located on any node in a VAXcluster system that has Volume Shadowing enabled.
- Clusterwide, homogeneous shadow-set maintenance functions.
- Ability to survive controller, disk, and media failures transparently.
- Shadowing functions that do not affect application I/O.

Applications and users read and write data to and from a shadow set using the same commands and program language syntax and semantics that are used for nonshadowed I/O operations. Volume Shadowed sets are managed and monitored using the same commands and utilities that are used for nonshadowed disks. The only difference is that access is through the virtual unit, not to individual devices.

SHDRIVER, the driver that controls the virtual unit functions, is described in Section 7.3.

For more detailed information on Volume Shadowing for OpenVMS, see the *Volume Shadowing for OpenVMS* manual.

7.2 Configurations

Volume Shadowing for OpenVMS does not depend on specific hardware in order to operate. All shadowing functions can be performed on VAX and Alpha systems running the OpenVMS operating system. Shadow set members must have the same physical geometry (that is, the same number of identical logical blocks [LBNs]) and members can be located anywhere in a VAXcluster system.

7.2.1 Hardware Supported

Volume Shadowing for OpenVMS does not depend on specific hardware to operate. All shadowing functions can be performed on any type of VAX or Alpha system using the OpenVMS operating system. Volume Shadowing requires a minimum of one VAX computer, one mass storage controller, and DSA (DIGITIAL Storage Architecture) or Small Computer System Interface (SCSI) disk drives.

See the most recent Volume Shadowing for OpenVMS *Software Product Descriptions* (SPD 27.29.*xx*) for additional information about hardware requirements.

7.2.2 Compatible Disk Drives and Volumes

Volume Shadowing requires compatibility among the physical units (disk drives and volumes) that form a shadow set. For instance:

- Units must have the same geometry, including the same number of sectors per track, the same number of tracks per cylinder, and the same number of cylinders per volume.
- Units must be Files–11 On-Disk Structure Level 2 (ODS-2) data disks.
- Units and controllers must conform to DSA and OpenVMS MSCP, or must be SCSI compliant.
- Units should not have hardware write protection enabled. Hardware write protection stops the Volume Shadowing software from maintaining identical volumes.

7.3 Driver Functions

This section describes the major virtual unit functions supported by SHDRIVER. In addition to the virtual unit functions described in this section, SHDRIVER supports all OpenVMS disk functions. SHDRIVER receives QIO operations from application programs and is a client of the disk class drivers DUDRIVER. Applications access the shadow set as they would access a standard OpenVMS disk.

Table 7–1 summarizes the major SHDRIVER functions. The subsections that follow describe these functions in detail.

Table 7–1 Functions of the Shadow Set Virtual Unit Driver

Function	Description
CRESHAD	Creates a virtual unit
ADDSHAD	Evaluates a physical member and adds members
COPYSHAD	Triggers and controls copy operations
REMSHAD	Removes a physical member
AVAILABLE	Virtual unit dissolution
SENSECHAR	Verifies shadow set status
READ	Directs I/O to a physical member
WRITE	Propagates a write operation to all physical members

7.3.1 CRESHAD

The CRESHAD function creates a virtual unit, and establishes a clusterwide lock. This function is internal and can be issued only by the MOUNT utility, from either DCL or the $MOUNT system service.

The function code is:

IO$_CRESHAD

IO$_CRESHAD takes the function modifier IO$M_EXISTS. When IO$M_EXISTS is specified, the virtual unit exists in the cluster and IO$_CRESHAD creates an identical, multimember set. If IO$M_EXISTS is not set, the shadow set does not exist yet and IO$_CRESHAD creates a single member shadow set.

IO$_CRESHAD also validates the shadow-set virtual unit, enables distributed locking protocols, and creates or updates the unit control block (UCB) and shadow set (SHAD) structures for the virtual unit.

The following are the device-dependent or function-dependent arguments for IO$_CRESHAD. (Note that these arguments are internal; they cannot be accessed by user programs.)

- P1—The address of the shadow-set virtual unit name string, or zero.
- P2—If P1 does not equal zero, this parameter is the size of the shadow set virtual unit name string. If P1 equals zero, this parameter is the unit number of the shadow-set virtual unit.
- P3—The storage control block (SCB) logical block number (LBN) from the first member listed in the MOUNT command.

IO$_CRESHAD can return the following status codes:

- SS$_NORMAL—Normal successful completion.
- SS$_ACCVIO—An access violation occurred because the shadow-set virtual unit name string is not readable.
- SS$_INSFMEM—Not enough memory exists to allocate the UCB or SHAD structure.
- SS$_IVDEVNAM—The unit number for the shadow set virtual unit is greater than 9999.

- SS$_INCSHAMEM—The specified shadow set member cannot be a member of the existing shadow set because it is the quorum disk or is of incompatible geometry.
- SS$_DUPINIT—The specified shadow set already exists.

7.3.2 ADDSHAD

The ADDSHAD function is an internal control function that validates the channel number and unit control block (UCB) address for a proposed shadow-set virtual unit member. This function also validates the specified copy-type information for the shadow-set virtual unit member and performs a clusterwide update. IO$_ADDSHAD then adds the member to the shadow set by updating the disk data structure and notifying other shadow set members.

The function code is:

IO$_ADDSHAD

IO$_ADDSHAD takes no function modifiers.

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

- P2—The channel number assigned to the proposed shadow set member
- P3—The copy type (copy or merge) of the member

IO$_ADDSHAD can return the following status codes:

- SS$_NORMAL—Normal successful completion.
- SS$_BADPARAM—The UCB is not a virtual unit UCB.
- SS$_ILLIOFUNC—The unit is not a shadow set virtual unit, or P2 points to a shadow set virtual unit.
- SS$_NOPRIV—The user has no access to the channel specified in P2.
- SS$_IVCHAN—The channel number specified in P2 is invalid.
- SS$_IVMODE—The copy-type mode specified in P3 is invalid.
- SS$_INSFMEM—Insufficient memory is available to allocate the shadow set member volume control block (VCB), and resource wait mode is disabled.
- SS$_INCSHAMEM—The physical unit cannot be added to the shadow set for one of the following reasons:
  - The shadow set is already fully populated.
  - The device being added is not an OpenVMS MSCP device.
  - The geometry or hardware properties of the physical unit do not match those of the other shadow set members.
  - The proposed shadow set member is the quorum disk.
  - The physical unit is already a member of another shadow set.

7.3.3 COPYSHAD

The COPYSHAD function triggers and controls copy operations. It should be issued only by the MOUNT utility or the shadow server process.

The function code is:

IO$_COPYSHAD

IO$_COPYSHAD takes the following function modifiers:

- IO$M_COPYOP—Requests a copy operation. This modifier is issued by the shadow server when the server requests a copy operation.
- IO$M_STEPOVER—Provides for protection of the volume storage control block (SCB) during copy operations.

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

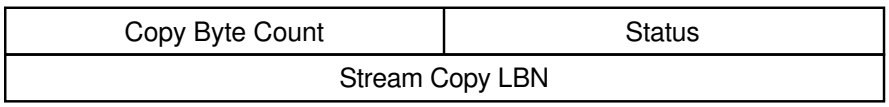
- P2—The request byte count
- P3—The starting LBN
- P4—The copy mode

IO$_COPYSHAD can return an SS$_ILLIOFUNC status for one of the following reasons:

- A COPY I/O command was issued by a process other than the shadow server.
- Another copy operation was attempted while a copy operation was already in progress.
- A COPY I/O command was issued while a copy operation was not in progress.

Figure 7–1 illustrates the I/O status block returned for copy operations.

Figure 7–1 I/O Status Block for Copy Operations



ZK–9699–GE

IO$_COPYSHAD receives copy initiation I/O request packets (IRPs) during a copy operation. IO$_COPYSHAD then performs the full or merge copy operation; upon completion, IO$_COPYSHAD receives IO$_COPYSHAD IRPs from the shadow server. A SS$_RESET message is returned if a copy-state reconciliation is needed because the copy initialization process has triggered a state change. The shadow server then issues an IO$_SENSECHAR function (see Section 7.3.6).

7.3.4 REMSHAD

The REMSHAD function removes a device from a shadow set virtual unit. The REMSHAD function can be issued only by the $DISMOUNT system service to a physical device.

The function code is:

IO$_REMSHAD

The IO$_REMSHAD function takes no modifiers.

IO$_REMSHAD verifies that the unit to be removed is a valid physical volume and a member of a shadow set. If either of these conditions is not true, SH$REMSHAD_FDT returns an SS$_ILLIOFUNC status. IO$_REMSHAD then removes the specified device from a shadow set by breaking the links between the device UCB and the other shadow-set data structures; clears device status fields; and removes references to the device from the device array in the SHAD structure. IO$_REMSHAD also updates the SCBs of disks that remain in the shadow set after changes are made to the local data structures.

If the shadow set has too few members to remove one, a SS$_BADPARAM status code is returned. Otherwise, SH$START_REMSHAD returns SS$_NORMAL to indicate a normal successful completion. Upon successful completion, IO$_REMSHAD notifies the remaining shadow set members.

7.3.5 AVAILABLE

The AVAILABLE function makes a virtual unit available by eliminating local control of the unit. AVAILABLE also makes the necessary changes to the local shadowing I/O database for disassembling the shadow set on the node, and releases cluster shadowing locks.

The function code is:

IO$_AVAILABLE

The IO$_AVAILABLE function takes no modifiers.

IO$_AVAILABLE first unloads the FDT (function decision table) for shadowing. If the specified unit is a member of a shadow set, or if IO$V_DISSOLVE is present for a physical unit, IO$_AVAILABLE returns an SS$_ILLIOFUNC status code. IO$_AVAILABLE then makes all changes to the local shadowing I/O database that are necessary to dismount the shadow set on the node.

7.3.6 SENSECHAR

The SENSECHAR function verifies the existence and membership status of a shadow set.

The function code is:

IO$_SENSECHAR

This function code takes the following function modifiers:

- IO$M_SHADOW
- IO$M_COPYOP

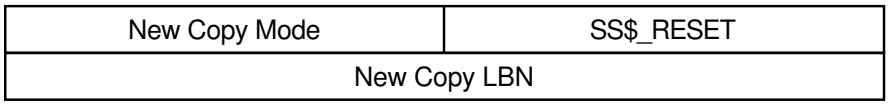
When the IO$M_SHADOW modifier is specified, the IO$_SENSECHAR verifies the presence of a properly prepared volume control block (VCB), initializes the VCB if necessary, and passes the request to the driver start I/O routine. If the IO$M_SHADOW modifier is not specified, the routine returns to the FDT processing loop. An SS$_BADPARAM status code is returned if the shadow-set virtual unit VCB is not present or is set up incorrectly.

IO$_SENSECHAR processing then begins. If specified, the modifiers IO$M_SHADOW and IO$M_COPYOP are used by the shadow server to determine whether or not to do any more copy operations before deallocating its resources.

IO$_SENSECHAR then ensures that the VCB address information in the SHAD data structure is up to date.

Finally, IO$_SENSECHAR determines whether any copy requests were previously suspended. A status code of SS$_NORMAL indicates that no copy operations were suspended. A status code of SS$_RESET indicates that copy operations are to continue; the LBN and mode data are supplied in the I/O status block, as shown in Figure 7–2.

Figure 7–2 I/O Status Block for Copy Information



ZK–9698–GE

7.3.7 Read and Write Functions

With just minor changes, the read and write functions for SHDRIVER operate the same as for the disk class driver (see Sections 2.4.1 and 2.4.2).

During an SHDRIVER read operation, the host directs the read to the member volume with the shortest path.

During a write operation, SHDRIVER directs the write to each member volume. The write operations for each member volume usually proceed in parallel; the virtual unit write operation terminates when all writes have completed. The write function for SHDRIVER takes the IO$M_VUEX_FC function modifier; this modifier should not be used by application programs.

The read and write SHDRIVER functions, as well as all user functions, are issued by user programs. All other SHDRIVER functions are invoked by MOUNT and DISMOUNT commands, or the $MOUNT and $DISMOUNT system services.

Remember that Volume Shadowing provides data availability by protecting against hardware problems or communication path problems that might cause a disk volume to be a single point of failure. If a write request is made to a shadow set, but the system fails before a completion status is returned from all of the shadow set members, it is possible that:

- All members might contain the new data
- All members might contain the old data
- Some members might contain new data and others might contain old data

When the system recovers, Volume Shadowing performs a merge or copy operation to ensure that the corresponding blocks on each shadow set member contain the same data (right or wrong). Thus, the goal here is not one of data correctness, but of data availability. Volume Shadowing is designed to make the data on all disks identical, then, if necessary, incorrect data can be reconciled either by the user reentering the data or by an application automatically employing database or journaling techniques.

For example, when used with Volume Shadowing, OpenVMS RMS journaling allows you to develop applications that can automatically recover from failures such as:

- Permanent loss of the path between a CPU data buffer containing the data being written and the disk being written to during a multiple block I/O

operation, Communication path loss can occur due to node failure or a failure of node-to-node communications.

- Failure of a CPU (such as a system crash, halt, power failure, or system shutdown) during a multiple block write I/O operation.
- Mistaken deletion of a file.
- Corruption of file system pointers.
- OpenVMS RMS file corruption due to a software error or incomplete bucket write operation to an indexed file.
- Cancellation of an in-progress multiple block write operation.

See the *Volume Shadowing for OpenVMS* manual for more information about shadowing merge and copy operations.

7.4 Error Processing

Shadow set recovery and repair are handled by volume processing, which replaces mount verification for shadow sets. The main difference in Phase II shadowing is that membership failure decisions are made by the VAX hosts. Device errors that result in inaccessibility of physical member units first utilize the class driver's connection walking algorithm. If that fails, a local decision is made on the shadow set membership. The rules are:

- If some, but not all, members of the set are accessible, then the local node sequentially adjusts the membership and notifies the other hosts.
- If no members are accessible, no modifications to the set membership are made.

There are two types of volume processing: active and passive. Active volume processing handles error processing on a local node. Triggered by a failed I/O operation, active volume processing also controls mount verification functions, member removal, and failover. Passive volume processing is triggered by lock messages or by a cluster event. Passive volume processing revalidates shadow set membership, ensures that the shadow set reflects changes made outside the shadow set, and handles the following functions:

- Member additions from other nodes
- Member removals from other nodes
- A new node mounting the shadow set
- A node dismounting the shadow set
- A system crash on a node that has the shadow set mounted

For more information, see the *Volume Shadowing for OpenVMS* manual.

8

Local Area Network (LAN) Device Drivers

This chapter describes the QIO interface of the local area network (LAN) drivers that control the devices listed in Table 8–1.

Table 8–1 Supported Communication Devices

Device	Driver
PMAD	ECDRIVER
‡Digital Ethernet EISA-bus Adapter (DE422)	ERDRIVER
‡Digital Ethernet EISA-bus Adapter (DE425)	ERDRIVER
‡Digital Ethernet ISA-bus Adapter (DE200)	ERDRIVER
‡Digital Ethernet ISA-bus Adapter (DE201)	ERDRIVER
‡Digital Ethernet ISA-bus Adapter (DE202)	ERDRIVER
DESVA	ESCRIVER
†Digital Ethernet BI-bus Network Adapter (DEBNA)	ETDRIVER
†Digital Ethernet BI-bus Network Interface (DEBNI)	ETDRIVER
‡Digital Quad Ethernet PCI-bus Adapter (DE436)	EWDRIVER
‡Digital Ethernet PCI-bus Adapter (DE434)	EWDRIVER
‡Digital Ethernet PCI-bus Adapter (TULIP)	EWDRIVER
Digital Ethernet XMI-bus Network Adapter (DEMNA)	EXDRIVER
†Second Generation Ethernet Chip (SGEC)	EZDRIVER
‡Third Generation Ethernet Chip (TGEC)	EZDRIVER
‡Digital FDDI Futurebus+ Adapter (DEFAA)	FADRIVER
Digital FDDI TURBOchannel Adapter (DEFZA)	FCDRIVER
Digital FDDI TURBOchannel Adapter (DEFTA)	FCDRIVER
†Digital FDDI Q–bus Adapter (DEFQA)	FQDRIVER
‡Digital FDDI EISA-bus Adapter (DEFEA)	FRDRIVER
‡Digital FDDI PCI-bus Adapter (DEFPA)	FWDRIVER
Digital FDDI XMI-bus Adapter (DEMFA)	FXDRIVER
‡Digital Token Ring TURBOchannel Adapter (DETRA)	ICDRIVER
‡Proteon PROnet-4/16 EISA NIC (DW300)	IRDRIVER
‡Proteon PROnet-4/16 ISA NIC (DW110)	IRDRIVER
†Digital Ethernet LSI UNIBUS Adapter (DELUA)	XEDRIVER

†VAX specific.
‡Alpha specific.

(continued on next page)

Table 8–1 (Cont.) Supported Communication Devices

Device	Driver
†Digital Ethernet UNIBUS Network Adapter (DEUNA)	XEDRIVER
†Digital Ethernet Q–bus Network Adapter (DEQNA)	XQDRIVER
†Digital Ethernet LSI Q–bus Adapter (DELQA)	XQDRIVER
†Digital Ethernet Q–bus Turbo Adapter (DEQTA)	XQDRIVER

†VAX specific.

Note

The DEQTA is the OpenVMS name for the DELQA-YM.

All drivers support Ethernet and Institution for Electrical and Electronic Engineers (IEEE) 802 Standards, except where otherwise indicated. Section 8.2.3 describes the specific IEEE 802 features supported by the drivers.

8.1 Relevant Terms

The following is a list of terms relevant to local area networks:

- CSMA/CD — Stands for carrier sense multiple access with collision detection. A technique that mediates demands by nodes on a single shared network channel; carrier sense determines whether the communications medium is already in use, while collision detection detects when more than one node is attempting to transmit.
- Ethernet — One of the earliest and the most common local area networks (LANs), Ethernet can refer to either a general LAN application (for example, Ethernet address) or to the specific CSMA/CD technology that implements the Intel, Xerox, and Digital intercompany Ethernet specifications.
- FDDI — The current fiber-optic LAN. FDDI (Fiber Distributed Data Interface) has advantages over Ethernet that include ten times the data rate and significant increases in LAN diameter.

Alpha

- Token Ring — The IEEE 802.5 standard token passing ring. ♦

8.2 LAN Characteristics

The LAN controllers are direct-memory-access (DMA) devices that, along with additional external hardware, implement the Ethernet, FDDI, or Token Ring specifications. A single LAN controller, which is a piece of equipment connected to the system bus, communicates with the local system through the system bus. This controller communicates with remote systems implementing the Ethernet, FDDI or Token Ring specifications through the communication medium. The Ethernet specification is described in *The Ethernet–Data Link Layer and Physical Layer Specification*. The FDDI specifications are available from ANSI. The Token Ring specifications are available from IEEE.

The Ethernet controllers use a single multiaccess channel with carrier sense and collision detection (CSMA/CD) to provide direct access from the processor to the Ethernet.

The FDDI controllers use a fiber-optic cable to provide direct access from the processor to the FDDI Token Ring. Note that FDDI networks and Ethernet networks can be combined to form a single extended LAN. This lets applications running on a system connected to FDDI communicate with applications that run on a system connected to Ethernet.

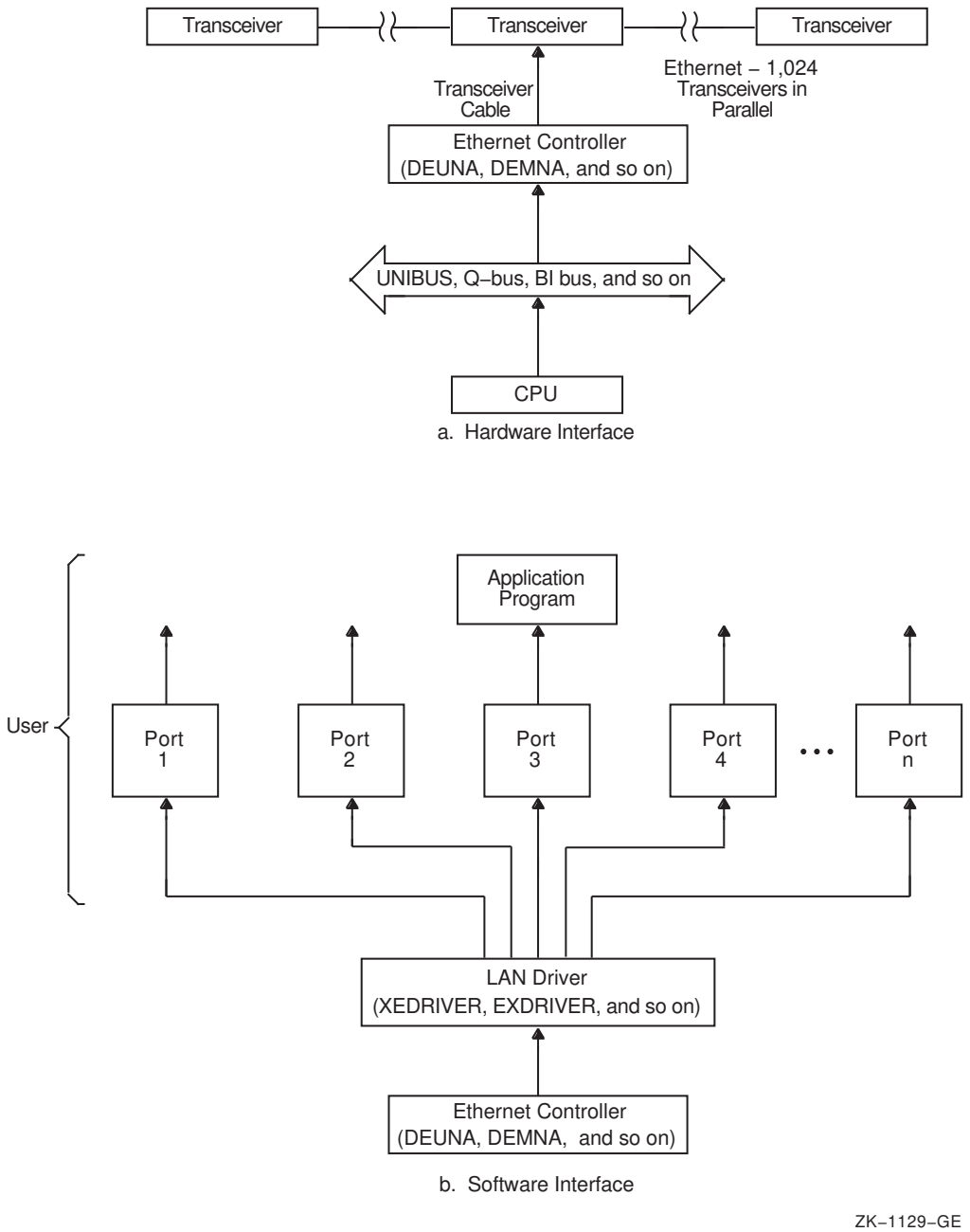
Alpha

On Alpha systems, the Token Ring controllers use either shielded or unshielded twisted pairs of wire to access the ring. Note that it is difficult to connect a Token Ring LAN directly bridged to any other type of LAN. However, routing protocols to other LANs work easily. ♦

A **port** in a LAN configuration consists of a protocol type, a service access point (SAP) or protocol identifier, and a controller. There are as many ports on a LAN controller as there are protocol types, SAPs, and protocol identifiers. Each port is independent of other ports running on the same LAN controller.

Application programs use the LAN driver's QIO interface to perform I/O operations to and from other nodes on the LAN. This chapter describes the QIO interface. Figure 8–1 shows the relationship of most Ethernet controllers to the processor and to the user application program.

Figure 8–1 Typical Ethernet Configuration



8.2.1 Driver Initialization and Operation

Digital recommends that you perform the following sequence to initialize and start a port on a LAN device driver:

1. Use the Assign I/O Channel ($ASSIGN) system service to assign I/O channels to one or more of the LAN device names and devices specified in Table 8–2. $ASSIGN creates a new unit control block (UCB), to which the channel for the port is assigned.

Table 8–2 LAN Device Names and Devices

Device Name	Device	Medium
ECc0:	PMAD	Ethernet
‡ERc0:	‡DE200	‡Ethernet
‡ERc0:	‡DE201	‡Ethernet
‡ERc0:	‡DE202	‡Ethernet
‡ERc0:	‡DE422	‡Ethernet
‡ERc0:	‡DE425	‡Ethernet
ESc0:	DESVA	Ethernet
ETc0:	DEBNA, DEBNI	Ethernet
‡EWc0:	‡TULIP	‡Ethernet
‡EWc0:	‡DE434	‡Ethernet
‡EWc0:	‡DE436	‡Ethernet
EXc0:	DEMNA	Ethernet
EZc0:	SGEC, TGEC	Ethernet
FAc0:	DEFAA	FDDI
FCc0:	DEFZA, DEFTA	FDDI
FQc0:	DEFQA	FDDI
‡FWc0:	‡DEFPA	‡FDDI
‡FRc0:	‡DEFEA	‡FDDI
FXc0:	DEMFA	FDDI
‡ICc0:	‡DETRA	‡Token Ring
‡IRc0:	‡DW110	‡Token Ring
‡IRc0:	‡DW300	‡Token Ring
XEc0:	DELUA, DEUNA	Ethernet
XQc0:	DEQNA, DELQA, DEQTA	Ethernet

‡Alpha specific.

Note

The *c* in the device name indicates the controller letter.

2. Start up the port with the set mode function and startup function modifier (see Section 8.8.3.1). You must supply the required P2 buffer parameters listed in Table 8–11.
3. Perform read, write, and sense mode operations as needed.
4. Shut down the port with the set mode function and shutdown function modifier (see Section 8.8.3.4).
5. Use the Deassign I/O Channel ($DASSGN) system service to deassign the I/O channel.

The sample programs described in Section 8.10.2 illustrate how to perform these procedures for Ethernet and IEEE 802 ports.

8.2.2 Ethernet Addresses

The LAN is a medium for creating a network; it is not a network by itself. The LAN controller and the local system constitute a node. Nodes on the LAN are identified by unique Ethernet addresses. A message can be sent to one, several, or all nodes on the LAN simultaneously, depending on the Ethernet address used. You do not have to specify the Ethernet address of your own node to communicate with other nodes on the same Ethernet. However, you do need to know the Ethernet address of the node with which you want to communicate.

8.2.2.1 Format of Ethernet Addresses

An Ethernet address is 48 bits in length. Ethernet addresses are represented by the Ethernet standard as six pairs of hexadecimal digits (six bytes), separated by hyphens (for example, AA-01-23-45-67-FF). The bytes are displayed from left to right in the order in which they are transmitted; bits within each byte are transmitted from right to left. In this example, byte AA is transmitted first; byte FF is transmitted last. (See the description of NMA$C_PCLI_PHA in Table 8–11, Section 8.8.3.1, for the internal representation of addresses.)

Upon application, IEEE assigns a block of addresses to a producer of LAN nodes. Thus, every manufacturer has a unique set of addresses to use. Normally, one address out of the assigned block of physical addresses is permanently associated with each controller (usually in read-only memory). This address is known as the hardware address of the controller. Each individual controller has a unique hardware address.

8.2.2.2 Ethernet Address Classifications

An Ethernet address can be a physical address of a single node or a multicast address, depending on the value of the low-order bit of the first byte of the address (this bit is transmitted first). Following are the two types of node addresses:

- Physical address—The unique address of a single node on a LAN. The least significant bit of the first byte of a physical address is 0. (For example, in physical address AA-00-03-00-FC-00, byte AA in binary is 1010 1010, and the value of the low-order bit is 0.)
- Multicast address—A multidestination address of one or more nodes on a given LAN. The least significant bit of the first byte of a multicast address is 1. (For example, in the multicast address 0B-22-22-22-22-22, byte 0B in binary is 0000 1011, and the value of the low-order bit is 1.)

Contrary to the Ethernet specification and the IEEE 802.3 Standard, the broadcast address (FF-FF-FF-FF-FF-FF) must be enabled as a multicast address in order to receive messages addressed to it.

8.2.2.3 Selecting an Ethernet Physical Address

The Digital interface to the LAN controllers allows you to set a physical address of the controller. The selection of the physical address of a LAN controller is different for CSMA/CD (Ethernet and 802.3) and FDDI.

For CSMA/CD, all users of the controller must agree on this address. The first user of the controller chooses the physical address; any additional users of the controller must specify either the same physical address, no physical address, or change the address (if allowed). When all channels to the controller are shut down, the next user to start a channel chooses the physical address. The controller's physical address is always chosen on the first successful startup when

there are no active ports. If the address is not chosen at this time, the controller's hardware address is used as the physical address.

For CSMA/CD, the Can Change Address parameter allows the physical address to be changed even though there are active users. If all current users of the controller have set the NMA$C_PCLI_CCA parameter to NMA$C_STATE_ON, then the physical address can be changed.

For FDDI, each port using a controller may specify its own unique physical address. Any combination of sharing of physical addresses is also allowed across the ports of an FDDI controller. For example, ports A, B, and C may use one unique physical address and ports D and E may use another unique address.

8.2.2.4 Ethernet Physical and Multicast Address Values

Digital multicast addresses are in the range AB-00-00-00-00-00 through AB-00-04-FF-FF-FF. The following are multicast addresses assigned for use in cross-company communications:

Value	Meaning
FF-FF-FF-FF-FF-FF	Broadcast
CF-00-00-00-00-00	Loopback assistance

The following Digital multicast addresses are assigned to be received by other nodes on the same Ethernet:

Value	Meaning
AB-00-00-01-00-00	Dump/load assistance
AB-00-00-02-00-00	Remote console
AB-00-00-03-00-00	Level 1 and Level 2 routers
AB-00-00-04-00-00	All end nodes
09-00-2B-02-00-00	Level 2 routers
AB-00-00-05-00-00 through AB-00-03-FF-FF-FF	Reserved for future use
AB-00-03-00-00-00	LAT
AB-00-04-00-00-00 through AB-00-04-00-FF-FF	For use by Digital customers for their own applications
AB-00-04-01-00-00 through AB-00-04-01-FF-FF	Local area VMScluster
AB-00-04-02-00-00 through AB-00-04-FF-FF-FF	Reserved for future use
09-00-2B-01-00-00	Digital Bridge management
09-00-2B-01-00-01	Digital Bridge hello multicast

8.2.2.5 Token Ring Functional Address Mapping (Alpha Only)

Alpha

On Alpha systems, except for the global broadcast address (FF-FF-FF-FF-FF-FF), Token Ring hardware does not support the 802 standard group LAN address mechanism. Instead, it uses **functional addresses**. These functional addresses are locally administered group addresses (multicast addresses). The first two bytes of the address are always 03-00 (canonical format), and the remaining four bytes contain a bit mask that specifies which of the 32 possible combination masks is being described.

Because most OpenVMS LAN applications use standard multicast addresses, a mechanism has been designed to map functional addresses to globally and locally administered multicast addresses. This allows applications to use the same multicast addresses that are used in the other LAN media.

Table 8–3 shows the mapping of the following OpenVMS Token Ring drivers:

Table 8–3 Address Mappings of Token Ring Drivers

Multicast Address	Functional Address	Bit-Reversed	Description
09-00-2B-00-00-04	03-00-00-00-02-00	C0:00:00:00:40:00	ISO 9542 All End-system Network Entites
09-00-2B-00-00-05	03-00-00-00-01-00	C0:00:00:00:80:00	ISO 9542 All Intermediate System Network Entities
CF-00-00-00-00-00	03-00-00-08-00-00	C0:00:00:10:00:00	Loopback Assistance
AB-00-00-01-00-00	03-00-02-00-00-00	C0:00:40:00:00:00	MOP Dump/Load
AB-00-00-02-00-00	03-00-04-00-00-00	C0:00:20:00:00:00	MOP Remote Console
AB-00-00-03-00-00	03-00-08-00-00-00	C0:00:10:00:00:00	DNA L1 Routers
09-00-2B-02-00-00	03-00-08-00-00-00	C0:00:10:00:00:00	DNA L2 Routers
09-00-2B-02-01-0A	03-00-08-00-00-00	C0:00:10:00:00:00	DECnet Phase IV—TRN—All Phase IV—TRN Routers
AB-00-00-04-00-00	03-00-10-00-00-00	C0:00:08:00:00:00	DNA End nodes
09-00-2B-02-01-0B	03-00-10-00-00-00	C0:00:08:00:00:00	Phase IV Prime Unknown
09-00-2B-00-00-07	03-00-20-00-00-00	C0:00:04:00:00:00	PCSA NETBIOS Emulatn
09-00-2B-00-00-0F	03-00-40-00-00-00	C0:00:02:00:00:00	Local Area Transport (LAT)
09-00-2B-02-01-04	03-00-80-00-00-00	C0:00:01:00:00:00	LAT Directory Service Solicit (to slave)
09-00-2B-02-01-07	03-00-00-02-00-00	C0:00:00:40:00:00	LAT Directory Service Solicit—X Service Class
09-00-2B-04-00-00	03-00-00-04-00-00	C0:00:00:20:00:00	LAST
09-00-2B-02-01-00	03-00-00-00-08-00	C0:00:00:00:10:00	DNA Naming Service Advertisement
09-00-2B-02-01-01	03-00-00-00-10-00	C0:00:00:00:08:00	DNA Naming Service Solicitation
09-00-2B-02-01-02	03-00-00-00-20-00	C0:00:00:00:04:00	DNA Time Service
03-00-00-00-00-01	03-00-00-00-00-01	C0:00:00:00:00:80	NETBUI Emulation

If an application needs to change or add mappings, QIOs exist for performing such operations. If the system or network manager has a requirement regarding mapping of the functional addresses, the LAN control program (LANCP) utility may be used to manage the mapping by running **sys$system:lancp**. The

following example maps the multicast address AB-01-01-01-02-03 to functional address 03-00-00-01-00-00 on Token Ring device ICA0:.

```
LANCP>SET DEVICE/MAP= -
_LANCP>(MULTICAST=AB-01-01-01-02-03,-
_LANCP>FUNCTIONAL=00-01-00-00) ICA0:
```

Note that it is possible for more than one multicast address to map to the same functional address. In all cases, the use of the functional address is associated with an individual application's protocol. ♦

8.2.3 Industry-Standard Support

CSMA/CD drivers support the following features:

- Ethernet and IEEE 802.3 packet format
- Physical layer identified as type 10Base5 (10-megabit/second baseband medium with maximum segment length of 500 meters), 10Base2 (ThinWire), and 10BaseT (twisted pair).

FDDI drivers support the following features:

- FDDI packet format
- Transmission and reception of frame control (FC) priority

Alpha

Token Ring drivers support the following features:

- IEEE 802.5 packet format ♦

All LAN drivers support the following features:

- 802.2 packet format
- IEEE 802.2 Class I service including the UI, XID and TEST commands, and the XID and TEST responses

  (Class II service must be provided by the user.)
- 6-byte destination and source address fields

  The IEEE 802.3 Standard states that the size of the destination and source addresses be 2 or 6 bytes, as decided by the manufacturer. The Digital LAN drivers and controllers only support 6-byte address fields.

Contrary to the IEEE 802.2 Standard, the Global DSAP (FF) must be enabled as a Group SAP to receive messages with the Global DSAP in the destination SAP field.

FDDI conforms to the ANSI Standards defined in the following documents:

- *ANSI X3.139-1987 FDDI Media Access Control (MAC)*
- *ANSI X3.148-1988 FDDI Physical Layer Protocol (PHY)*
- *ANSI X3.166-1990 FDDI Physical Layer Medium Dependent (PMD)*

8.3 Procedures for Configuring ISA Devices

Alpha

There are two ways to configure a ISA LAN device on OpenVMS. They are as follows:

- Create an entry in the SYS$MANAGER:ISA_CONFIG.DAT file
- Use 'isacfg' from the console prompt (>>>)

If you choose to create an entry in the SYS$MANAGER:ISA_CONFIG.DAT file, see SYS$MANAGER:ISA_CONFIG.TEMPLATE for an example. To use 'isacfg' from your console prompt, see the hardware documentation associated with your system for more information.

Regardless of which configuration method you choose, you should become familiar with the concepts listed in Table 8–4. See your LAN hardware documentation for configuring information on how to set the jumper settings for those features in Table 8–4.

Table 8–4 ISA Configuring Concepts

Concept	Explanation
ISA Slot Number	Also called node. The ISA slot number of the device. *OpenVMS Alpha Device Support: Developer's Guide* describes how to number ISA slots.
IRQ	Interrupt request line. Used to establish the interrupt level. Boards support IRQs between 1 and 15. Since OpenVMS does not support shared IRQs, every ISA device must have its own IRQ value reserved for it by the console ISACFG utility.
Port Address	I/O base control status registers (CSR) address. Boards usually have I/O addresses associated with their CSR locations. These locations must not be used by any other device on the system.
Memory Address	Also called shared memory. This address range is used to share memory resident on the adapter card between the host CPU and the LAN device on the adapter card. If the device uses shared memory, the I/O addresses for accessing this memory must be reserved. These locations must not be used by any other device on the system.
DMA Channel	If the device performs direct memory access (DMA), either slave or bus mastering, a DMA channel is required. Channels are numbered 1 through 7, but channel 4 is always reserved for the system. See your adapter documentation for DMA channels supported by the device.

8.3.1 OpenVMS LAN Devices Requiring Configuration

The DE202 Ethernet and DW110 Token Ring devices require configuring. The following information and examples show how this is done.

8.3.1.1 DE202 Ethernet

The DE202 is a shared memory Ethernet device. To configure the jumper settings on the board, see the adapter documentation. Example 8–1 and Example 8–2 show how to configure the OpenVMS software to use the DE202. The following two examples illustrate a configuration of:

- Slot 1
- IO Base at %x300
- IRQ 5
- Shared memory at %xd0000 with length of %x10000

Example 8–1 Using the 'isacfg' at Console Prompt

```
>>> isacfg -slot 1 -etyp 1 -ena 1 -irq0 5 -iobase0 %x300
          -membase0 %xd0000 -memlen0 %x1000 -handle "DE20" -mk
```

The command ("-mk") makes an isacfg entry for an ISA device at slot 1. It is a Single port type of device (-etyp 1). The -handle parameter tells the operating system what type of device it is.

Example 8–2 Using the SYS$MANAGER:ISA_CONFIG.DAT Entry

```
[ERA0]
NAME=ER
DRIVER=SYS$ERDRIVER
NODE=1                        ; plugged into ISA Option slot 1
IRQ=5
PORT=(300:f)                  ; 15 bytes starting at 300
MEM= (D0000:10000)            ; 64 bytes starting at D0000
```

8.3.1.2 DW110 Token Ring

The DW110 is a bus mastering DMA device on the ISA bus. In addition to setting up the ISA I/O parameters, you may configure ring speed (4 or 16 Mbits) and media (UTP or STP). By using LANCP you can also configure ring speed and media during system startup. Example 8–3 and Example 8–4 show how to configure the OpenVMS software to use the DW110. The following two examples illustrate a configuration of:

- Slot 4
- IRQ 10
- DMA channel 7
- Base %x4e20
- Shielded Twisted Pair (STP)
- Ring speed of 16

Example 8–3 Using the 'isacfg' at Console Prompt

```
>>> isacfg -slot 4 -etyp 1 -ena 1 -irq0 %xa -dmachan0 7
          -iobase0 %x4e20 -handle "DW11,STP,16" -mk
```

The command ("-mk") makes an isacfg entry for an ISA device at slot 4. It is a Single port type of device (-etyp 1). The -handle parameter tells the operating system that this is a DW110, that STP media is to be used, and the ring speed is 16.

Example 8–4 Using the SYS$MANAGER:ISA_CONFIG.DAT entry

```
[IRA0]
NODE=4
NAME=IR
DRIVER=SYS$IRDRIVER
IRQ=A
DMA=7
PORT=(4E20:20)
USER_PARAM="STP,16"
```

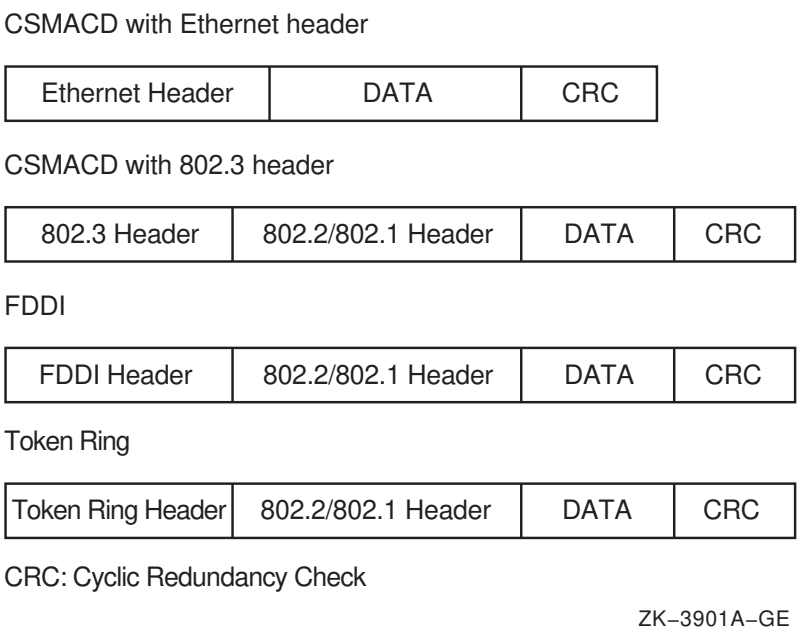
For more detailed information, see the *OpenVMS Alpha Device Support: Developer's Guide*. ♦

8.4 Frame Formats

Because two media (CSMA/CD and FDDI) are supported by OpenVMS, there are some differences in frame formats. The following sections describe the similarities and differences in these frame formats. Despite differences, the QIO interface to the LAN drivers is designed to allow applications to run over the different media with few changes to the application.

The frame formats available in the LAN media are shown in Figure 8–2.

Figure 8–2 LAN Frame Formats



Note that CSMA/CD provides two frame formats and the FDDI provides one frame format. The 802.1 header is an optional extension to the 802.2 header.

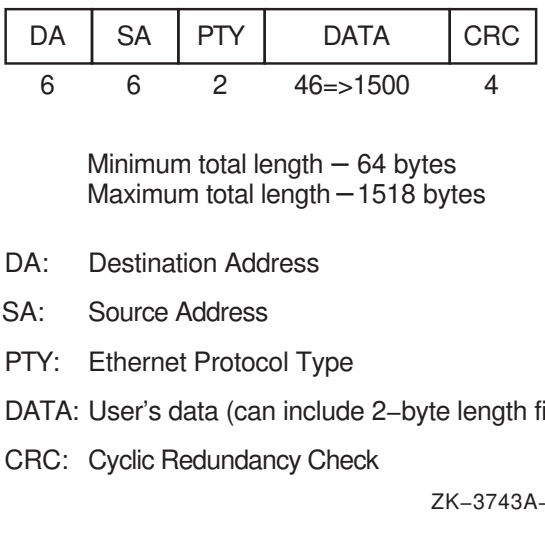
8.4.1 CSMA/CD Frames

There are two headers for CSMA/CD frames.

- Ethernet header
- IEEE 802.3 header

Figure 8–3 illustrates a CSMA/CD frame with an Ethernet header.

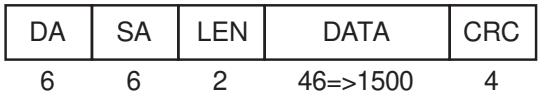
Figure 8–3 CSMA/CD Frame with Ethernet Header



The Ethernet header consists of the DA, SA, and PTY fields. Ethernet frames must be at least 64 bytes in length, which means that the minimum data length is 46 bytes. Applications select Ethernet format by specifying NMA$C_LINFM_ ETH (the default) as the value for NMA$C_PCLI_FMT in their P2 characteristics buffer. If the amount of actual data to be transmitted is less than 46 bytes, the CSMA/CD drivers transmit extra bytes of zero after the application data.

Figure 8–4 illustrates a CSMA/CD frame with an IEEE 802.3 header.

Figure 8–4 CSMA/CD Frame with IEEE 802.3 Header



DA: Destination Address

SA: Source Address

LEN: The length of data portion only. It can be less than 46 if the user supplied less than 46 bytes of data, but the frame is then padded to meet minimum length requirements.

DATA: User's data

CRC: Cyclic Redundancy Check

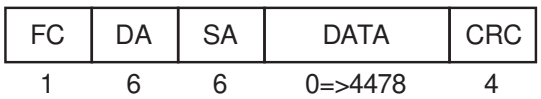
ZK–3745A–GE

The IEEE 802.3 format is similar to the Ethernet format, except the PTY field is replaced by the LEN field.

8.4.2 FDDI Frames

Figure 8–5 illustrates the format of FDDI frames.

Figure 8–5 FDDI Frame Format



FC: Frame Control contains a "priority" field that can be used to determine if the frame originated on the FDDI or on the Ethernet.

DA: Destination Address

SA: Source Address

DATA: User's data

CRC: Cyclic Redundancy Check

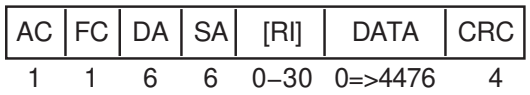
ZK–3742A–GE

The FDDI header consists of the FC, DA, and SA fields.

8.4.3 Token Ring Frames (Alpha Only)

Figure 8–6 illustrates the format of Token Ring frames.

Figure 8–6 Token Ring Frame Format (Alpha Only)



AC: Access Control contains priority for the frame.
FC: Frame Control contains the type of frame.
DA: Destination Address
SA: Source Address
RI: Optional Routing Information. Only valid with packets that are source routed.
DATA: User's data
CRC: Cyclic Redundancy Check

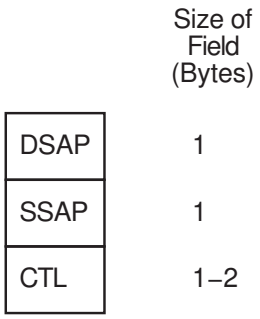
ZK–6679A–GE

8.4.4 802.2/802.1 Headers

The 802.2 header follows the 802.3 header in a CSMA/CD frame and follows the FDDI header in an FDDI frame.

The 802.2 header is illustrated in Figure 8–7.

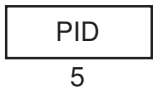
Figure 8–7 802.2 Header



ZK–4799–GE

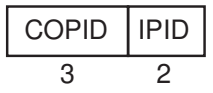
This 802.2 header is followed by the 802.1 header illustrated in Figure 8–8 if the DSAP field is the SNAP SAP (AA hex), the SSAP field is the SNAP SAP, and the CTL field is UI (03 hex).

Figure 8–8 802.1 Header



The PID field consists of two subfields: the company portion of the PID and the implementation-specific portion of the PID (see Figure 8–9).

Figure 8–9 802.1 Header Subfields

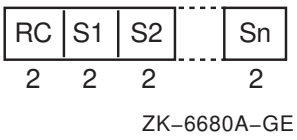


8.4.5 Token Ring Source Routing Header (Alpha Only)

Alpha

Figure 8–10 details the field of the source routing header.

Figure 8–10 Source Routing Field (Alpha Only)



The routing control (RC) contains the information about how the packet is to be routed.

The Sn contains the segment identifier for the hop. Each segment identifier contains a ring number and bridge number used in the hop. ♦

8.5 Format Parameter

Each port (or user) of an Ethernet/FDDI controller either specifies a value for the format characteristic of the port or assumes the default. The format characteristic has three valid values: Ethernet, 802, and 802E. This section describes the actual frame formats for each medium for each format parameter value.

For a format parameter value of NMA$C_LIMFM_ETH (the default), the frame formats are shown in Figure 8–11. Note that a name given to the FDDI frame with the Ethernet format is a mapped Ethernet frame.

Figure 8–11 Frames with Ethernet Format

For CSMA/CD

DA	SA	PTY	DATA	CRC
6	6	2	46–1500	4

For FDDI

FC	DA	SA	DSAP	SSAP	CTL	COPID	IPID	DATA	CRC
1	6	6	1	1	1	3	2	0–4470	4

For Token Ring

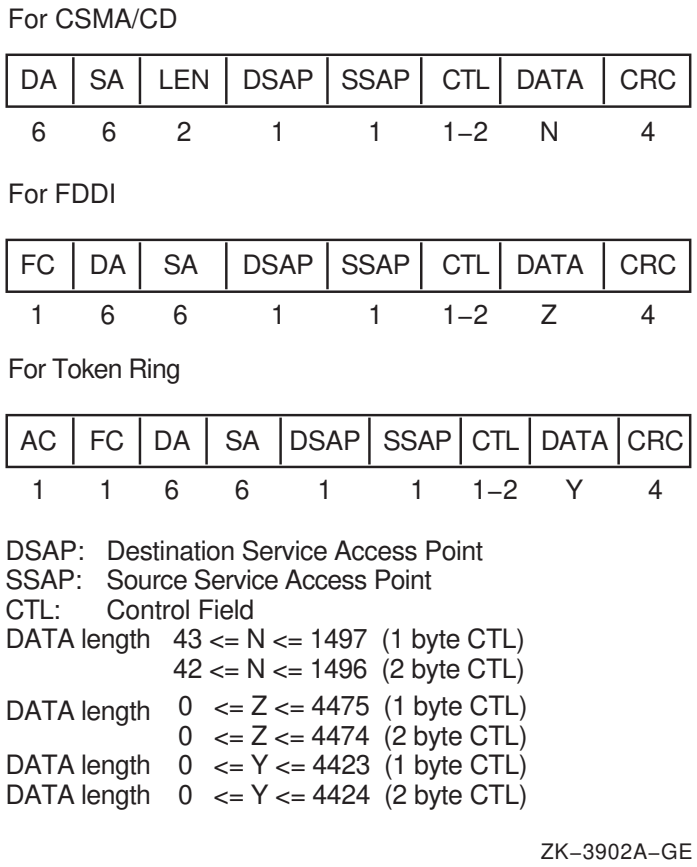
AC	FC	DA	SA	DSAP	SSAP	CTL	COPID	IPID	DATA	CRC
1	1	6	6	1	1	1	3	2	0–4418	4

DSAP: SNAP SAP (hex AA)
SSAP: SNAP SAP
CTL: UI (hex 03)
COPID: Mapped Ethernet (00–00–00)
IPID, PTY: Protocol Type

ZK–3905A–GE

For a format parameter value of NMA$C_PCLI_802, the frame formats are shown in Figure 8–12.

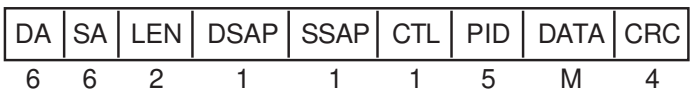
Figure 8–12 Frames with 802 Format



For a format parameter value of NMA$C_PCLI_802E, the frame formats are shown in Figure 8–13.

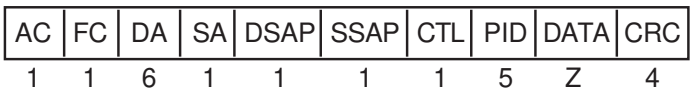
Figure 8–13 Frames with 802E Format

For CSMA/CD



For FDDI

For Token Ring



DSAP: SNAP SAP (hex AA)
SSAP: SNAP SAP
CTL: UI (hex 03)
PID: User's Protocol Identifier
DATA length 38 <= M <= 1492
DATA length 0 <= Y <= 4470
DATA length 0 <= Z <= 4418

ZK–3903A–GE

8.6 Features of Formats

The Digital Ethernet controllers can transmit and receive both Ethernet and 802.2/802.3 packets. Each port on a controller is able to transmit and receive either Ethernet or 802 packets. Ethernet and 802 ports can be assigned on the same controller at the same time.

The Digital FDDI controllers can transmit and receive FDDI frames. There is a mapped Ethernet frame format that is comparable to the Ethernet packets sent and received by the Ethernet controllers.

At the time each port on the controller is started, one of three packet formats can be specified: Ethernet (default), standard 802 (referred to as 802 packet format), and extended 802. If no format is specified, the default format is used.

Each port on the controller must be unique on that controller. For each packet format, there is a parameter that distinguishes the port from all other ports with the same packet format. For Ethernet packet format ports, the 2-byte protocol type parameter defines the port. For 802 packet format ports, the 1-byte SAP defines the port. For extended 802 format ports, the 5-byte protocol identifier defines the port.

Sections 8.6.1, 8.6.2, and 8.6.3 describe the three packet formats and the characteristics unique to each format.

8.6.1 Ethernet Packet Format

The Ethernet format has a few unique characteristics. The packet assembly /disassembly (PAD) parameter allows a receiving node to determine the amount of real data in the frame. Ethernet format also allows protocol type sharing. The following sections describe these features.

8.6.1.1 Ethernet Protocol Types

Every Ethernet frame has a 2-byte protocol-type field. This field is used to determine the port to which a packet belongs. Protocol types are independent of addresses; unique protocol designations can be assigned by Xerox Corporation. Whenever an Ethernet user at a particular station starts an Ethernet port, that user must specify the protocol type to be used on that port. Messages sent over that port always have the protocol type attached to them by the device driver, and messages received with that protocol type are delivered to the starter of that port. The Digital protocol types are in the ranges 60-00 through 60-09 and 80-38 through 80-42. Valid protocol types are in the range 05-DD through FF-FF.

Following are the cross-company protocol types:

Value	Meaning
08-00	IP protocol (ARPAnet)
08-06	Address resolution protocol (ARPAnet)
90-00	Loopback

The Digital protocol types are as follows:

Value	Meaning
60-00	Reserved for Digital
60-01	Dump/load assistance
60-02	Remote console
60-03	DECnet
60-04	LAT
60-05	Diagnostics
60-06	For use by Digital customers for their own applications
60-07	Local area VAXcluster
60-08	Reserved for Digital
60-09	Reserved for Digital
80-38	Digital Bridge
80-39 through 80-3B	Reserved for Digital
80-3C	Name server
80-3D	Reserved for Digital
80-3E	Time server
80-3F	Reserved for Digital
80-40	BIOS
80-41	LAST

Value	Meaning
80-42	Reserved for Digital

8.6.1.2 Ethernet Packet Padding

This section describes the packet assembly/disassembly (PAD) parameter NMA$C_PCLI_PAD, which is used only in the Ethernet packet format.

All CSMA/CD frames must be at least 64 bytes in length. This includes the Ethernet header, the user data, and the CRC. If the user data, CRC, and Ethernet header together are less than 64 bytes, null padding bytes are inserted between the user data and the CRC to make a 64-byte packet. This packet padding cannot be turned off.

The PAD parameter allows the LAN drivers to place a data-size field in the packet between the standard header and the user data. If padding is on (NMA$C_ STATE_ON is specified) , a 2-byte length field is inserted after the Protocol Type field and before the user data.

If the PAD parameter is off (NMA$C_STATE_OFF is specified), Ethernet packets have the following characteristics:

- Packets transmitted are padded with null bytes as needed (CSMA/CD only).
- Packets transmitted do not include the size field.
- The length of user data in the packets received is always between 46 and 1500 bytes for CSMA/CD, and 0 to 4470 for FDDI. For example, if a 10-byte packet is transmitted, it is received as 46 bytes because the driver cannot determine the amount of user data in the packet—only the amount of user data plus padded null bytes.

If the PAD parameter is on (NMA$C_STATE_ON is specified), Ethernet packets have the following characteristics:

- Packets transmitted are padded with null bytes as needed (CSMA/CD only).
- Packets transmitted include the size field.
- The length of user data in the packets received is always between 0 and 4468 bytes. The driver uses the size field to determine the amount of user data in the packet.

8.6.1.3 Protocol Type Sharing

Protocol types are usually nonshareable. The problems inherent in sharing a protocol type include the multiplexing and demultiplexing of messages to and from remote nodes. However, the protocol access parameter (NMA$C_PCLI_ACC) allows a protocol type to be opened in either of two shareable modes: shared-default (NMA$C_ACC_SHR) and shared-with-destination (NMA$C_ACC_LIM). The LAN drivers also provide the nonshareable exclusive mode (NMA$C_ACC_ EXC). (See Table 8–11.) The rules and requirements for using each mode are as follows:

- The exclusive mode is the default if no access mode is supplied as a P2 buffer parameter. This mode of operation does not allow the protocol to be shared by other users. Any attempt to start up another protocol of the same type results in an error status of SS$_BADPARAM.

- The shared-with-destination mode is a protocol type/destination address pairing that allows multiple users to share a protocol type and to communicate with a different node.

  For a given shared protocol type, there can be many ‘‘shared-with-destination’’ users; each user communicates with a different destination address. Any attempt to start a port with a destination address that is in use results in an error status of SS$_BADPARAM.

  When a ‘‘shared-with-destination’’ user passes the set mode P2 buffer, the buffer must contain a destination address in the NMA$C_PCLI_DES parameter. This destination address is used as the destination address in all messages transmitted, and the user receives messages only from this address.

- The shared-default mode is the default user of a shared protocol type. There can be only one such user for each shared protocol type. A ‘‘shared-default’’ user does not have to exist if a protocol type is shared, but there can be no more than one such user per shared protocol type.

  The ‘‘shared-default’’ user receives all messages for the shared protocol type, but not for any of the ‘‘shared-with-destination’’ users. The ‘‘shared-default’’ user also receives all messages matching both the shared protocol type and any multicast address enabled by the ‘‘shared-default’’ user.

  The ‘‘shared-default’’ user can only transmit to multicast addresses and physical addresses that are not enabled by any of the ‘‘shared-with-destination’’ users sharing the same protocol type.

  If there is no ‘‘shared-default’’ user of a protocol type, incoming messages from nodes not among the ‘‘shared-with-destination’’ users for that protocol type are ignored.

8.6.2 IEEE 802 Packet Format

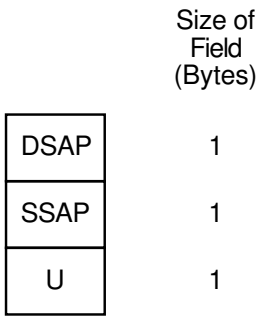
The IEEE 802 packet formats accepted for a port depend on the service enabled on that port. All 802 packet formats have an 802.2 header. The service on the port determines the valid values for the 802.2 fields.

When a port is started, the NMA$C_PCLI_SRV parameter in the P2 buffer selects the service on that port. A value of NMA$C_LINSR_CLI specifies Class I service and a value of NMA$C_LINSR_USR specifies user-supplied service (the default).

8.6.2.1 Class I Service Packet Format

For Class I service, only three packet formats are transmitted and received: UI, XID, and TEST. Figure 8–14 shows the 802.2 header format for Class I service.

Figure 8–14 Class I Service 802.2 Header



The control field for an 802 packet is always an unnumbered control field. The unnumbered control field, which is always 1 byte in length, is passed by the P4 argument of the write QIO and can be one of the following binary values:

- UI command (00000011)

  This is the unnumbered information command. It is the method used to transmit data from one user to another and is the most widely used control field value.

  The UI command can be specified by using NMA$C_CTLVL_UI.

- XID command (101p1111)

  This is the exchange identification command. It is used to convey information about the port. The ‘‘p’’ bit is the poll bit and can be either 0 or 1. This command can be specified by using NMA$C_CTLVL_XID for a ‘‘0’’ poll bit or NMA$C_CTLVL_XID_P for a ‘‘1’’ poll bit.

- XID response (101f1111)

  The XID response is a response to an XID command. The ‘‘f’’ bit is the final bit and will match the poll bit from the XID command.

- TEST command (111p0011)

  The TEST command is used to test a connection. The ‘‘p’’ bit is the poll bit and can be either 0 or 1. This command can be specified by using NMA$C_ CTLVL_TEST for a ‘‘0’’ poll bit or NMA$C_CTLVL_TEST_P for a ‘‘1’’ poll bit.

- TEST response (111f0011)

  The TEST response is a response to a TEST command. The ‘‘f’’ bit is the final bit and will match the poll bit from the TEST command.

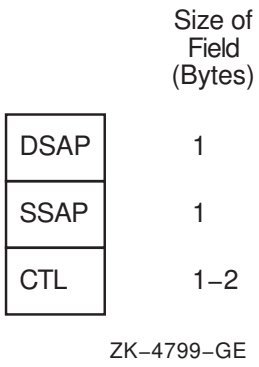
An 802 format port with Class I service is allowed to transmit UI, XID, and TEST commands. An 802 format port with Class I service is allowed to receive UI commands and XID and TEST responses.

See the IEEE 802.2 Standard for more information on these control field values and response messages.

8.6.2.2 User-Supplied Service Header Format

Figure 8–15 shows the 802.2 header format for user-supplied service.

Figure 8–15 User-Supplied Service 802.2 Header

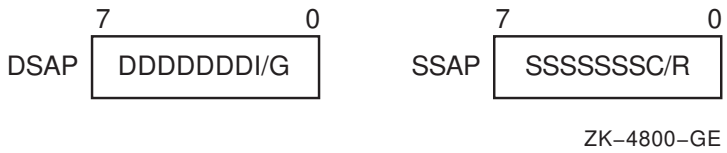


The user provides the control field values, which are documented in the IEEE 802.2 Standard. The user-supplied packet format is the generic packet format as specified in the IEEE 802.2 Standard. Class I packets (see Section 8.6.2.1) are a subset of this generic packet format. Therefore, if the control field value of the user-supplied packet is UI, XID, or TEST, the packet is the same as a Class I packet. Note that Class II packets, as defined in the IEEE 802.2 Standard, include the UI, XID, and TEST command/response formats.

8.6.2.3 Service Access Point (SAP) Use and Restrictions

The IEEE 802.2 Standard places restrictions on both user SAPs and source SAPs (SSAPs). All SAPs are 8 bits long. Figure 8–16 shows the format of desination SAPs (DSAPs) and SSAPs.

Figure 8–16 DSAP and SSAP Format



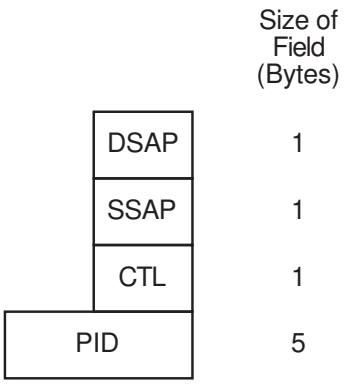
Definition of the least significant bit depends on whether the SAP is a source SAP (SSAP) or a destination SAP (DSAP). For a DSAP field, the least significant bit distinguishes group SAPs (bit 0 = 1) from individual SAPs (bit 0 = 0). For an SSAP field, the least significant bit distinguishes commands (bit 0 = 0) from responses (bit 0 = 1). Because these two bits are located at the same bit position within the SAP field, a group SAP cannot be used as an SSAP. If this were allowed, a group SAP would be interpreted as an individual SAP with the command/response bit set to 1, thus implying a response. The IEEE 802.2 Standard reserves for its own definition all SAP addresses with the second least significant bit set to 1. You should use these SAP values for their intended purposes, as defined in the IEEE 802.2 Standard.

Up to four group SAPs can be enabled on each 802 port. The group SAPs enabled on a controller do not have to be unique for each port; for example, two 802 format ports can have the same group SAP enabled. This allows a single packet coming into the controller to be duplicated and passed to each port on the controller that has the group SAP enabled—assuming the packet has a DSAP value that is a group SAP. If the received packet has an individual SAP for a DSAP, the packet goes to at most one port.

8.6.3 IEEE 802 Extended Packet Format

802E format uses the 802.2 and 802.1 headers, as shown in Figure 8–17.

Figure 8–17 802 Extended Header



ZK–5791–GE

For an 802E packet format, the DSAP and SSAP fields are always set to the SNAP SAP (AA hex). The SNAP SAP value is a special SAP value reserved for 802 extended format packets. The SNAP SAP value distinguishes an 802 packet from an 802 extended packet. The only valid control field value for 802 extended packets is UI (unnumbered information).

8.6.3.1 Protocol Type PID Sharing (Alpha Only)

Alpha

On Alpha systems, the 802E format allows user's protocol identifier (PID) sharing. PIDs are usually nonshareable. The problems inherent in sharing a PID include the multiplexing and demultiplexing of messages to and from remote nodes. However, the protocol access parameter (NMA$C_PCLI_ACC) allows a PID to be opened in either of two shareable modes: shared-default (NMA$C_ACC_SHR) and shared-with-destination (NMA$C_ACC_LIM). The LAN drivers also provide the nonshareable exclusive mode (NMA$C_ACC_EXC). (See Table 8–11.) The rules and requirements for using each mode are as follows:

- The exclusive mode is the default if no access mode is supplied as a P2 buffer parameter. This mode of operation does not allow the protocol to be shared by other users. Any attempt to start up another protocol of the same type results in an error status of SS$_BADPARAM.
- The shared-with-destination mode is a PID/destination address pairing that allows multiple users to share a PID and to communicate with a different node.

For a given shared protocol type, there can be many ‘‘shared-with-destination’’ users; each user communicates with a different destination address. Any attempt to start a port with a destination address that is in use results in an error status of SS$_BADPARAM.

When a ‘‘shared-with-destination’’ user passes the set mode P2 buffer, the buffer must contain a destination address in the NMA$C_PCLI_DES parameter. This destination address is used as the destination address in all messages transmitted, and the user receives messages only from this address.

- The shared-default mode is the default user of a shared PID. There can be only one such user for each shared PID. A ‘‘shared-default’’ user does not have to exist if a PID is shared, but there can be no more than one such user per shared PID.

  The ‘‘shared-default’’ user receives all messages for the shared PID, but not for any of the ‘‘shared-with-destination’’ users. The ‘‘shared-default’’ user also receives all messages matching both the shared PID and any multicast address enabled by the ‘‘shared-default’’ user.

  The ‘‘shared-default’’ user can only transmit to multicast addresses and physical addresses that are not enabled by any of the ‘‘shared-with-destination’’ users sharing the same PID.

  If there is no ‘‘shared-default’’ user of a PID, incoming messages from nodes not among the ‘‘shared-with-destination’’ users for that PID are ignored. ♦

8.7 LAN Device Information

You can obtain information on controller characteristics by using the Get Device /Volume Information ($GETDVI) system service. (See the *OpenVMS System Services Reference Manual*.) $GETDVI returns controller characteristics when you specify the item code DVI$_DEVCHAR. Table 8–5 lists these characteristics, which are defined by the $DEVDEF macro and in the file SYS$LIBRARY:DEVDEF.H.

Table 8–5 Ethernet Controller Device Characteristics

Characteristic	Meaning
	Static Bits (Always Set)
DEV$M_AVL	Device is available
DEV$M_IDV	Input device
DEV$M_NET	Network device
DEV$M_ODV	Output device

DVI$_DEVTYPE and DVI$_DEVCLASS return the device type and device class names, which are defined by the $DCDEF macro and in the file SYS$LIBRARY:DCDEF.H.

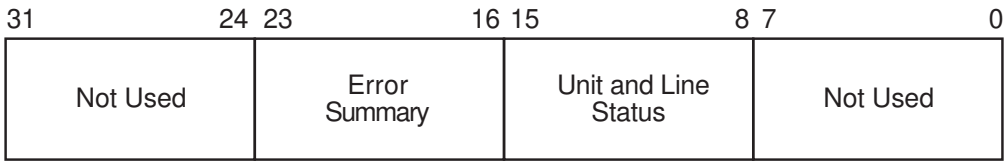
Ethernet Controller	Device	
	Type	**Class Name**
DEBNA	DT$_ET_DEBNA	DC$_SCOM

Ethernet Controller	Device	
	Type	Class Name
DEBNI	DT$_ET_DEBNI	DC$_SCOM
DEFAA	DT$_FA_DEFAA	DC$_SCOM
DEFEA	DT$_FR_DEFEA	DC$_SCOM
DEFQA	DT$_FQ_DEFQA	DC$_SCOM
DEFTA	DT$_FC_DEFTA	DC$_SCOM
‡DEFPA	‡DT$_FW_DEFPA	‡ DC$_SCOM
DEFZA	DT$_FC_DEFZA	DC$_SCOM
‡DETRA	‡DT$_IC_DETRA	DC$_SCOM
DELQA	DT$_XQ_DELQA	DC$_SCOM
DELUA	DT$_DELUA	DC$_SCOM
DEMFA	DT$_FX_DEMFA	DC$_SCOM
DEMNA	DT$_EX_DEMNA	DC$_SCOM
DEQNA	DT$_DEQNA	DC$_SCOM
DEQTA	DT$_XQ_DEQTA	DC$_SCOM
DESVA	DT$_ES_LANCE	DC$_SCOM
DEUNA	DT$_DEUNA	DC$_SCOM
‡DE200	‡DT$_ER_DE422	‡ DC$_SCOM
‡DE201	‡DT$_ER_DE422	‡ DC$_SCOM
‡DE202	‡DT$_ER_DE422	‡ DC$_SCOM
‡DE422	‡DT$_ER_DE422	‡ DC$_SCOM
‡DE425	‡DT$_ER_TULIP	‡ DC$_SCOM
‡DE434	‡DT$_EW_DE435	‡ DC$_SCOM
‡DE436	‡DT$_EW_DE435	‡ DC$_SCOM
‡DW110	‡DT$_IR_DW300	‡ DC$_SCOM
‡DW300	‡DT$_IR_DW300	‡ DC$_SCOM
PMAD	DT$_EC_PMAD	DC$_SCOM
SGEC	DT$_EZ_SGEC	DC$_SCOM
‡TGEC	‡DT$_EZ_TGEC	‡ DC$_SCOM
‡TULIP	‡DT$_EW_TULIP	‡ DC$_SCOM

‡Alpha specific.

DVI$_DEVBUFSIZ returns the maximum message size. The maximum send or receive message size depends on the packet format and whether padding (NMA$C_PCLI_PAD) is enabled (see Sections 8.8.1 and 8.8.2). DVI$_DEVDEPEND returns the unit and line status bits and the error summary bits in a longword field as shown in Figure 8–18.

Figure 8–18 DVI$_DEVDEPEND Returns



ZK–5932–GE

Table 8–6 lists the status values and their meanings. These values are defined by the $XMDEF macro. XM$M_STS_ACTIVE is set when the port is started. XM$M_STS_BUFFAIL and XM$M_STS_TIMO are dynamically set and cleared by the LAN driver.

Table 8–6 Ethernet Controller Unit and Line Status

Status	Meaning
XM$M_STS_ACTIVE	Port is active.
XM$M_STS_BUFFAIL	Attempt to allocate a system receive buffer failed.
XM$M_STS_TIMO	Timeout occurred.

The error summary bits are set when an error occurs. They are read-only bits. If an error is fatal, the Ethernet port is shut down. Table 8–7 lists the error summary bit values and their meanings.

Table 8–7 Error Summary Bits

Error Summary Bit	Meaning
XM$M_ERR_FATAL	Hardware or software error occurred on controller.

8.8 LAN Function Codes

The LAN drivers can perform logical, virtual, and physical I/O operations. The basic functions are read, write, set mode, set characteristics, sense mode, and sense characteristics. Table 8–8 lists these functions and their codes. The following sections describe these functions in greater detail.

Table 8–8 LAN I/O Functions

Function Code	Arguments	Type[1]	Function Modifiers	Function
IO$_READLBLK[3]	P1,P2,[P5]	L	IO$M_NOW	Read logical block.
IO$_READVBLK[3]	P1,P2,[P5]	V	IO$M_NOW	Read virtual block.
IO$_READPBLK[3]	P1,P2,[P5]	P	IO$M_NOW	Read physical block.

[1] V = virtual, L = logical, P = physical (There is no functional difference in these operations.)

[3] On OpenVMS Alpha, P1, and P5 support 64-bit addresses.

(continued on next page)

Table 8–8 (Cont.) LAN I/O Functions

Function Code	Arguments	Type[1]	Function Modifiers	Function
IO$_WRITELBLK[4]	P1,P2,[P4],P5	L	IO$M_RESPONSE	Write logical block.
IO$_WRITEVBLK[4]	P1,P2,[P4],P5	V	IO$M_RESPONSE	Write virtual block.
IO$_WRITEPBLK[4]	P1,P2,[P4],P5	P	IO$M_RESPONSE	Write physical block.
IO$_SETMODE	P1,[P2],P3[2]	L	IO$M_CTRL IO$M_STARTUP IO$M_SHUTDOWN IO$M_ATTNAST IO$M_SET_MAC IO$M_UPDATE_MAP IO$M_ROUTE	Set controller characteristics and controller state for subsequent operations.
IO$_SETCHAR	P1,[P2],P3[2]	P	IO$M_CTRL IO$M_STARTUP IO$M_SHUTDOWN IO$M_ATTNAST IO$M_SET_MAC IO$M_UPDATE_MAP IO$M_ROUTE	Set controller characteristics and controller state for subsequent operations.
IO$_SENSEMODE	[P1],[P2]	L	IO$M_CTRL IO$M_SENSE_MAC IO$M_SHOW_MAP IO$M_SHOW_ROUTE	Sense controller characteristics and return them in specified buffers.
IO$_SENSECHAR	[P1],[P2]	P	IO$M_CTRL IO$M_SENSE_MAC IO$M_SHOW_MAP IO$M_SHOW_ROUTE	Sense controller characteristics and return them in specified buffers.

[1]V = virtual, L = logical, P = physical (There is no functional difference in these operations.)

[2]The P1 and P3 arguments are only for attention AST QIOs.

[4]On OpenVMS Alpha, P1, P4, and P5 support 64-bit addresses.

Although the LAN device drivers do not differentiate among logical, virtual, and physical I/O functions (all are treated identically), you must have the required privilege to issue the request. (Logical I/O functions do not require I/O privilege.)

8.8.1 Read

Read functions directly transfer data from a packet received from another port on the Ethernet into the virtual memory address space of the user process. The operating system provides the following function codes:

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

Received messages are buffered in system memory and then copied to the user's buffer when a read operation is performed.

The read functions take the following device- or function-dependent arguments:

- P1—The starting virtual address of the buffer that is to receive data. On OpenVMS Alpha systems, P1 can be a 64-bit address.
- P2—The size of the receive buffer in bytes.

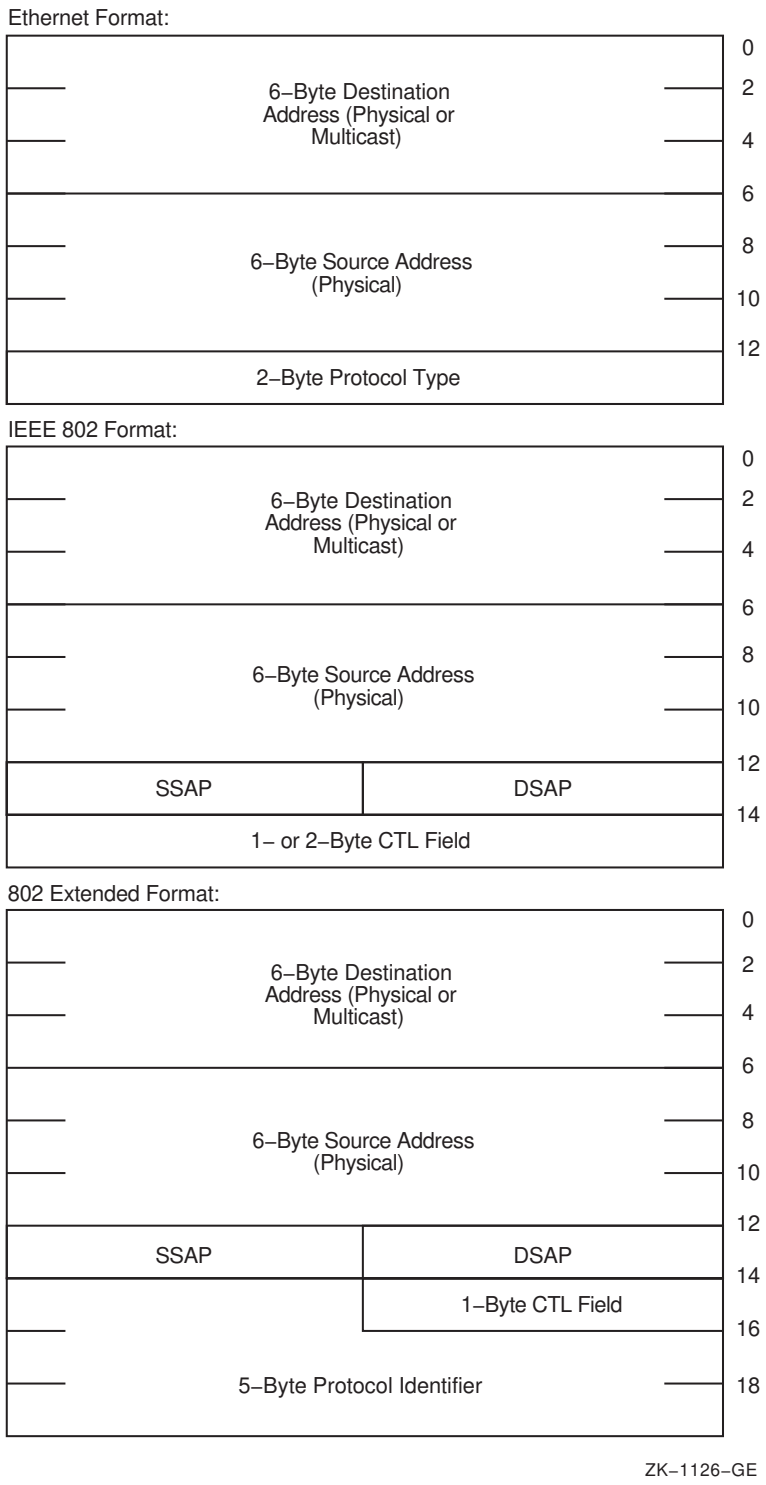
- P5—The address of a buffer where the LAN driver returns packet header information. This is an optional argument. The information returned depends on the packet format enabled with the set mode QIO. The size of the buffer must be 14 bytes for an Ethernet format packet, 16 bytes for an IEEE 802 format packet, and 20 bytes for an 802 extended format packet. Note that the information returned is not the entire packet header but the header information less any length or size fields. The IOSB, if specified, is where the packet length information is returned. For FDDI, if received access control (RAC) is on, then 1 byte must be added to these sizes.

  Alpha

  On Alpha systems, Token Ring requires that this buffer be at least 54 bytes long due to a possible variable length source routing header. ♦

  If NMA$C_PCLI_PRM (see Table 8–11) is enabled, the P5 buffer must be 20 bytes for CSMA/CD and 21 bytes for FDDI. Figure 8–19 shows the format of the three buffers. On OpenVMS Alpha systems, P5 can be a 64-bit address.

Figure 8–19 Read Function P5 Buffer



The P1 and P2 arguments must always be specified; the P5 argument is optional. However, if P5 is not specified, you will be unable to determine the source of the received message.

If the size of the user data in a receive message is larger than the value of the NMA$C_PCLI_BUS parameter, the message is not given to the user, even if there is sufficient space in the user's receive buffer.

If the size of the user data in a receive message is larger than the size specified in P2 (and less than or equal to the value of the NMA$C_PCLI_BUS parameter), the P1 buffer is filled and SS$_DATAOVERUN is returned in the I/O status block.

Table 8–9 lists the maximum user data sizes that can be received.

Table 8–9 Maximum User Data Sizes

Packet Format	CSMA/CD	FDDI	‡Token Ring
Ethernet format without padding	1500	4470	‡4418
Ethernet format with padding	1498	4468	‡4416
802 format with 1-byte CTL field	1497	4475	‡4423
802 format with 2-byte CTL field	1496	4474	‡4422
802E format	1492	4470	‡4418

‡Alpha specific.

For 802 format packets, the P5 buffer always contains the DSAP and SSAP in the bytes at offset 12 and 13. The next one or two bytes (offsets 14 and 15) following the SSAP contain the control field value. For Class I service, the control field value is always 1 byte in length and will always be placed in the byte at offset 14 of this buffer. For user-supplied service, you have to determine the length of the control field value according to the IEEE 802.2 Standard.

Alpha

On Alpha systems with Token Ring, if received access control (RAC) is on, the first byte of the P5 buffer contains the frame control (FC) field. ♦

For FDDI, if RAC is on, the first byte of the P5 buffer contains the FC field.

The read functions can take the following function modifier:

IO$M_NOW—Complete the read operation immediately with a received message (if no message is currently available, return a status of SS$_ ENDOFFILE in the I/O status block).

8.8.2 Write

Write functions provide for the direct transfer of data from the virtual memory address space of the user process to the communications medium. The operating system provides the following function codes:

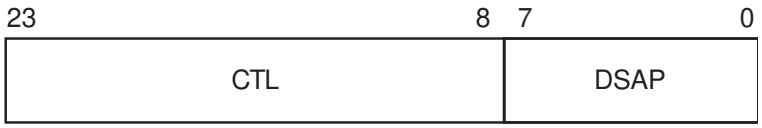
- IO$_WRITELBLK—Write logical block
- IO$_WRITEVBLK—Write virtual block
- IO$_WRITEPBLK—Write physical block

Transmitted messages are copied from the buffer of the requesting process to a system buffer for transmission.

The write function takes the following device- or function-dependent arguments:

- P1—The starting virtual address of the buffer containing the data to be transmitted. On OpenVMS Alpha systems, P1 can be a 64-bit address.
- P2—The size of the buffer in bytes.

- P4—The address of a quadword that points to a buffer that contains the DSAP and CTL field values (optional). (See Section 8.6.2.3.) The first longword is the buffer length; the second longword is the address of the buffer. This argument is used only for ports with the 802 packet format. The format of the buffer is:



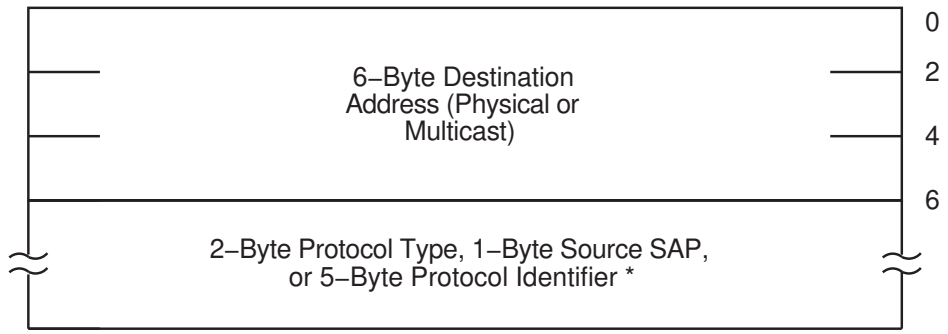
ZK–4801–GE

  On OpenVMS Alpha systems, P4 can be a 64-bit address.

- P5—The address of a 6-byte buffer that contains the destination address (either physical or multicast). For FDDI, if XFC is specified as zero on startup, the first byte of the P5 buffer contains the low-order 3 bits of the FC field to be transmitted. On OpenVMS Alpha systems, P5 can be a 64-bit address.

  If the device is in promiscuous mode (NMA$C_PCLI_PRM; see Table 8–11), you must pass a larger buffer with additional information positioned after the destination address. For Ethernet packet format, the buffer must be 8 bytes with the 2-byte protocol type following the destination address. For 802 packet format, the buffer must be 7 bytes with the 1-byte source SAP following the destination address. For 802 extended packet format, the buffer must be 11 bytes with the 5-byte protocol identifier following the destination address. The individual Source SAP cannot be a group SAP or the SNAP SAP. Figure 8–20 shows the format of the P5 buffer. For FDDI with XFC specified as zero on startup, 1 byte must be added to these sizes for the FC field.

Figure 8–20 Write Function P5 Buffer



*Only if the channel is in promiscuous mode.

ZK–1211–GE

Table 8–10 lists the maximum user data sizes that can be specified by P2.

Table 8–10 Maximum Message Sizes

Packet Format	CSMA/CD	FDDI	‡Token Ring
Ethernet format without padding	1500	4470	‡4418
Ethernet format with padding	1498	4468	‡4416
802 format with 1-byte CTL field	1497	4475	‡4423
802 format with 2-byte CTL field	1496	4474	‡4422
802E format	1492	4470	‡4418

‡Alpha specific.

If P2 specifies a message size larger than that allowed, the driver returns the status SS$_IVBUFLEN in the I/O status block.

If the P4 buffer is specified, it must be at least 3 bytes long. The first byte is always the DSAP; the next 2 bytes are used to determine the CTL field value. The DSAP value cannot be the SNAP SAP.

The CTL field value is either a 1-byte or 2-byte value. If the two least significant bits of the low-order byte of the CTL field contain the bit values 11, just the low-order byte of the CTL field is used as the CTL field value. Otherwise, both bytes of the CTL field are used as the CTL field value.

If the driver uses only the low-order byte of the CTL field, you still must pass at least a 3-byte buffer. In this case, the driver uses the low-order byte of the CTL field and ignores the high-order byte.

If Class I service is enabled, only 1-byte CTL field values can be passed. If user-supplied service is enabled, then both 1- and 2-byte CTL field values are valid. If Class I service is enabled, the CTL field value must be one of the three command values: UI, XID, or TEST.

Regarding 802 ports, you can receive packets for the SAP enabled with the IO$_SETMODE or IO$_SETCHAR QIOs and can transmit packets destined for a different SAP. This is similar to an Ethernet port receiving packets for one protocol type and transmitting packets with a different protocol type (which is not possible with the current Ethernet $QIO interface). It is expected that most 802 format applications will want to process only receive packets from a source SAP that matches the SAP enabled on their port. To do this, the read function (see Section 8.8.1) has been enhanced to return the source SAP to you. To verify that the source SAP of an incoming packet matches the SAP enabled on the port, you need only match the source SAP returned by the read function with the SAP enabled on the port.

The write functions can take the following function modifier:

IO$M_RESPONSE—Transmit a response packet (sets the low-order bit in the SSAP field). This allows users with user-supplied service enabled to respond to certain 802 format command packets. IO$M_RESPONSE can be specified only when you have the 802 packet format enabled. The 802 packet format ports, with Class I service enabled, result in an error if you attempt to transmit a response message with a CTL field value of unnumbered information (UI).

8.8.3 Set Mode and Set Characteristics

The operating system provides the following two function codes:

IO$_SETMODE
IO$_SETCHAR

Other than the privilege check, these two function codes are treated the same by the LAN drivers. This section refers to the IO$_SETMODE function code only, even though applications can use either function code.

The set mode function code is used to perform many different functions. These different functions are distinguished by the modifiers set with the function code. The LAN drivers support the following set mode requests:

- IO$_SETMODE!IO$M_CTRL—set or modify port attributes
- IO$_SETMODE!IO$M_CTRL!IO$M_STARTUP—set port attributes and start port
- IO$_SETMODE!IO$M_SET_MAC—set medium attributes
- IO$_SETMODE!IO$M_CTRL!IO$M_SHUTDOWN—shutdown port
- IO$_SETMODE!IO$M_ATTNAST—enable attention AST

Alpha

The LAN drivers also support the following set mode requests for Token Ring:

- IO$M_SETMODE!IO$M_UPDATE_MAP—Update functional address mapping table
- IO$M_SETMODE!IO$M_ROUTE— Update source routing cache table ♦

The following sections describe these functions in detail.

8.8.3.1 Set Controller Mode

Once a port is created using the $ASSIGN system service, you can set the port attributes and start the port using the requests listed in the previous section. Note that in most cases only IO$_SETMODE!IO$M_CTRL!IO$M_STARTUP is issued because it sets the port attributes and starts the port with one request. IO$_SETMODE!IO$M_CTRL is most often used to modify port attributes after the port has been started.

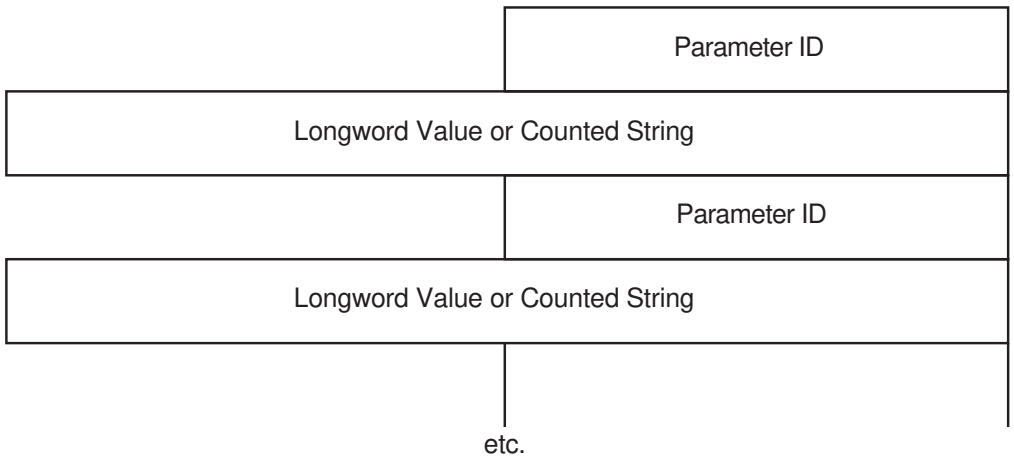
If the function modifier IO$M_STARTUP is specified, the LAN port is started. If IO$M_STARTUP is not specified, the specified characteristics are modified.

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

P2—The address of a quadword descriptor for an extended characteristics buffer. The first longword of the descriptor is the buffer length; the second longword is the address of the buffer. The P2 argument is optional.

The P2 buffer consists of a series of 6-byte or counted string entries. The first word of each entry contains the parameter identifier (ID) of an attribute, followed by either a longword that contains one of the (binary) values that can be associated with the parameter ID or a counted string. Counted strings consist of a word that contains the size of the character string followed by the character string. Figure 8–21 shows the format for this buffer.

Figure 8–21 P2 Extended Characteristics Buffer



ZK–1177–GE

Table 8–11 is an alphabetic listing of the parameter IDs and values that can be specified in the P2 buffer. These parameter IDs are applicable to all LAN controllers, except where otherwise noted. The $NMADEF macro defines these values. The $NMADEF macro is included in the macro library SYS$LIBRARY:LIB.MLB. (Table 8–12 lists the parameters that can be used with each of the packet formats, and indicates which are required, which are optional, and which generate the SS$_BADPARAM error.)

If the status SS$_BADPARAM is returned in the first word of the I/O status block, the second longword contains the parameter ID of the parameter in error.

Table 8–11 P2 Attributes

Parameter ID	Meaning
NMA$C_PCLI_ACC	†Protocol access mode. This optional parameter determines the access mode for the protocol type. NMA$C_PCLI_ACC is valid only for ports using Ethernet packet format. ‡NMA$C_PCLI_ACC is valid for ports using Ethernet and 802E. One of the following values can be specified: NMA$C_ACC_EXC — Exclusive mode (default) NMA$C_ACC_SHR — Shared-default user mode NMA$C_ACC_LIM — Shared-with-destination mode †Section 8.6.1.3 provides a description of protocol type sharing. ‡Section 8.6.3.1 provides a description of protocol type PID sharing. NMA$C_PCLI_ACC is passed as a longword value.

†VAX specific.
‡Alpha specific.

(continued on next page)

Table 8–11 (Cont.) P2 Attributes

Parameter ID	Meaning
NMA$C_PCLI_BFN	Number of receive buffers to preallocate (default = 1). NMA$C_PCLI_BFN can have a maximum value of 255. This optional parameter is specified on a per-port basis. NMA$C_PCLI_BFN is passed as a longword value. NMA$C_PCLI_BFN represents the number of receive messages the LAN driver will hold for a port when the port has no read QIOs posted to the driver.
NMA$C_PCLI_BUS	Maximum allowable port receive data size, that is, message length (default = 512 bytes). NMA$C_PCLI_BUS can have a maximum value of 4478. This optional parameter is specified on a per-port basis. It is passed as a longword value. Any message received for this port that is larger than this parameter value is not used to complete a read QIO.
NMA$C_PCLI_CCA	Can change address. This optional parameter enables applications to start before DECnet starts. DECnet may attempt to set the physical address of the controller when it starts. CSMA/CD devices support only one physical address, and so all applications that are using the same device must also use the same physical address. If applications that do not use the DECnet address start before DECnet, DECnet is not able to start on that controller unless the other applications that have already started have all specified NMA$C_PCLI_CCA to be ON. This parameter is not applicable to FDDI because FDDI devices can run with more than one physical address. However, no error is returned if this parameter is supplied for FDDI devices. The application receives no indication whatsoever that the physical address has changed. This parameter is passed as a longword. One of the following values can be specified: NMA$C_STATE_ON — The physical address can be changed. NMA$C_STATE_OFF — The physical address cannot be changed (default).
NMA$C_PCLI_CON[1]	Controller mode. This optional parameter determines whether transmit packets are to be looped back at the controller. This parameter has no effect on Token Ring devices. One of the following values can be specified: NMA$C_LINCN_NOR — Normal mode (default) NMA$C_LINCN_LOO — Loopback mode The only messages looped back are those acceptable to the controller as receive messages, that is, those messages that possess at least one of the following characteristics: • Matching physical address (see Section 8.2.2) • Matching multicast address (see Section 8.2.2) • Promiscuous mode (NMA$C_PCLI_PRM) is in the ON state • Destination address is a multicast address and all multicasts are enabled (NMA$C_PCLI_MLT is in the ON state) NMA$C_PCLI_CON affects all ports on a single controller. It is passed as a longword value.

[1]If the LAN controller is active and you do not specify this parameter, the parameter defaults to the current setting. If the LAN controller is not active, this parameter defaults to the default value indicated.

(continued on next page)

Table 8–11 (Cont.) P2 Attributes

Parameter ID	Meaning
	For the DELUA, DEBNA, DEBNI, DEQTA, PMAD, DEMNA, and DESVA, the following list shows the maximum amount of user data that can be looped: Ethernet format without padding — 18 bytes Ethernet format with padding — 16 bytes 802 format with 1-byte CTL field — 15 bytes 802 format with 2-byte CTL field — 14 bytes 802 extended format–10 bytes When the DEUNA is in loopback mode, the driver always enables echo mode (NMA$C_PCLI_EKO is in the ON state).
NMA$C_PCLI_CRC[1]	Cyclic redundancy check (CRC) generation state for transmitted messages (optional). One of the following values can be specified: NMA$C_STATE_ON — Controller generates a CRC (default). NMA$C_STATE_OFF — Controller does not generate a CRC. NMA$C_PCLI_CRC affects all ports on a single controller. There is no effect on checking a receive message's CRC (it is always checked). NMA$C_PCLI_CRC is passed as a longword value. If NMA$C_PCLI_CRC is turned off, all users of the controller must supply the 4-byte CRC value for all messages transmitted. The CRC is passed at the end of the P1 transmit buffer; the additional 4 bytes are included in the size of the P1 buffer. The CRC value is not checked for correctness. For the DEQNA, the DELQA, and the Token Ring devices, the NMA$C_PCLI_CRC parameter cannot be turned off.
NMA$C_PCLI_DES	Shared protocol destination address. Passed as a counted string that consists of a modifier word (NMA$C_LINMC_SET or NMA$C_LINMC_CLR) followed by a 6-byte (48-bit) physical destination address. The size of the counted string must always be 8. NMA$C_PCLI_DES only has meaning when protocol access (NMA$C_PCLI_ACC) is defined as shared-with-destination mode (NMA$C_ACC_LIM). The destination address specified must be a physical address—not a multicast address—and it must be unique among all ports sharing the same protocol. NMA$C_PCLI_DES is required when the access mode is defined as ''shared-with-destination.'' †NMA$C_PCLI_DES should not be specified on a port where the 802 or 802E packet format is selected (NMA$C_PCLI_FMT is set to NMA$C_LINFM_802 or NMA$C_LINFM_802E). For 802 packet format, the concept of shared protocol type is handled by using group SAPs. ‡NMA$C_PCLI_DES should not be specified on a port where the 802 packet format is selected (NMA$C_PCLI_FMT is set to NMA$C_LINFM_802). For 802 packet format, the concept of shared protocol type is handled by using group SAPs. †Section 8.6.1.3 provides a description of protocol type sharing. ‡Section 8.6.3.1 provides a description of protocol type PID sharing.

[1]If the LAN controller is active and you do not specify this parameter, the parameter defaults to the current setting. If the LAN controller is not active, this parameter defaults to the default value indicated.

†VAX specific.

‡Alpha specific.

(continued on next page)

Table 8–11 (Cont.) P2 Attributes

Parameter ID	Meaning
NMA$C_PCLI_EKO[1]	Echo mode. Applicable only to the DEUNA device driver. If echo mode is on, transmitted messages are returned to the sender. This optional parameter controls the condition of the half-duplex bit in the DEUNA mode register. One of the following values can be specified: NMA$C_STATE_ON — Echoes transmit messages NMA$C_STATE_OFF — Does not echo transmit messages (default) If NMA$C_STATE_ON is specified, the only transmitted messages echoed are those acceptable to the DEUNA as receive messages, that is, those messages that have at least one of the following characteristics: • Matching physical address (see Section 8.2.2) • Matching multicast address (see Section 8.2.2) • Promiscuous mode (NMA$C_PCLI_PRM) is in the ON state • Destination address is a multicast address and all multicasts are enabled (NMA$C_PCLI_MLT is in the ON state) If the DEUNA is placed in loopback mode (NMA$C_LINCN_LOO is specified in the NMA$C_PCLI_CON parameter), the driver enables echo mode. NMA$C_PCLI_EKO affects all ports on a single controller. It is passed as a longword value.
NMA$C_PCLI_FMT	Packet format. This optional parameter specifies the packet format as either Ethernet, IEEE 802, or 802 extended. This characteristic is passed as a longword value and affects single ports on a single controller. One of the following values can be specified: NMA$C_LINFM_ETH — Ethernet packet format (default) NMA$C_LINFM_802 — 802 packet format NMA$C_LINFM_802E — 802 extended packet format NMAC_PCLI_PTY, NMAC_PCLI_ACC, and NMA$C_PCLI_DES should only be specified on those ports where the Ethernet packet format (NMA$C_LINFM_ETH) is selected. NMAC_PCLI_SRV, NMAC_PCLI_SAP, and NMA$C_PCLI_GSP should only be specified on those ports where the 802 packet format (NMA$C_LINFM_802) is selected. NMA$C_PCLI_PID should only be specified on those ports where the 802 extended packet format (NMA$C_LINFM_802E) is selected.

[1]If the LAN controller is active and you do not specify this parameter, the parameter defaults to the current setting. If the LAN controller is not active, this parameter defaults to the default value indicated.

(continued on next page)

Table 8–11 (Cont.) P2 Attributes

Parameter ID	Meaning
NMA$C_PCLI_GSP	Group SAP. This is an optional parameter if the 802 packet format is selected (NMA$C_PCLI_FMT is set to NMA$C_LINFM_802). If the Ethernet or 802 extended packet format is selected, NMA$C_PCLI_GSP cannot be specified. Group SAPs can be shared among multiple ports on the same controller. If the 802 packet format is selected, NMA$C_PCLI_GSP defines up to four 802 group SAPs that are to be enabled for matching incoming packets to complete read operations on this port. By default, no group SAPs are enabled. NMA$C_PCLI_GSP is passed as a longword value and is read as four 8-bit unsigned integers. Each integer must be either a group SAP or zero. To enable a single group SAP on a port, you need only specify the group SAP value to be enabled in one of the four integers and place a value of zero in the three remaining integers. To disable group SAPs on the port, you need only place a value of zero in all four integers and issue the QIO. If this characteristic is correctly specified, any group SAPs that were previously enabled on the port are now replaced by the SAPs specified by the current request.
NMA$C_PCLI_ILP[1]	Internal loopback mode. This optional parameter places the device in internal loopback mode (not for the DEUNA, DEQNA, or DELQA devices). This parameter has no effect for Token Ring devices. One of the following values can be specified: NMA$C_STATE_ON — Internal loopback mode NMA$C_STATE_OFF — Not in internal loopback mode (default) If NMA$C_STATE_ON is specified, the NMA$C_PCLI_CON parameter must be in loopback (NMA$C_LINCN_LOO) mode. When the controller is in loopback mode (generally for testing), it can loop packets in external loopback or internal loopback. This parameter places the controller in one of these loopback modes. NMA$C_PCLI_ILP is passed as a longword value and affects all ports on the controller.
NMA$C_PCLI_MCA	Multicast address (optional). Passed as a counted string that consists of a modifier word followed by a list of 6-byte (48-bit) multicast addresses. The value specified in the modifier word determines whether the addresses are set or cleared. If NMA$C_LINMC_CAL is specified, all multicast addresses in the list are ignored. The following mode values can be specified in the low byte of the modifier word: NMA$C_LINMC_SET — Set the multicast addresses. NMA$C_LINMC_CLR — Clear the multicast addresses. NMA$C_LINMC_CAL — Clear all multicast addresses. The driver filters all multicast addresses on a per-port basis. Therefore, only messages received with the port's physical address or the multicast addresses enabled on the port are used to complete the user's read operations. Note that each LAN controller supports a limited number of multicast addresses. If this limit is exceeded, the LAN driver enables the ''accept all multicast'' feature on the controller and all multicast packets on the LAN must be filtered by the LAN driver. This may cause a minor performance loss. NMA$C_PCLI_MCA is specified on a per-port basis.

[1]If the LAN controller is active and you do not specify this parameter, the parameter defaults to the current setting. If the LAN controller is not active, this parameter defaults to the default value indicated.

(continued on next page)

Table 8–11 (Cont.) P2 Attributes

Parameter ID	Meaning
NMA$C_PCLI_MLT	Multicast address state. This optional parameter instructs the controller hardware whether to accept all multicast addresses for this port. One of the following values can be specified: NMA$C_STATE_ON — Accept all multicast addresses. NMA$C_STATE_OFF — Do not accept all multicast addresses (default). NMA$C_PCLI_MLT can be enabled on more than one port. It only affects those ports on which it is enabled. NMA$C_PCLI_MLT allows you to receive all multicast address packets that also match the port's protocol type, SAP, or protocol identifier. Generally, you enable only your individual set of multicast addresses using the NMA$C_PCLI_MCA parameter, and leave the NMA$C_PCLI_MLT parameter in the off state. There could be a minor performance loss when the NMA$C_PCLI_MLT parameter is in the ON state because the LAN driver may have to process all multicast addresses on the medium; the number of multicast addresses on the line determines the amount of processing required. The NMA$C_PCLI_MLT parameter is passed as a longword value.
NMA$C_PCLI_PAD	Use message size field on transmit and receive messages (optional). One of the following values can be specified: NMA$C_STATE_ON — Insert message size field (default) NMA$C_STATE_OFF — No size field NMA$C_PCLI_PAD affects only the protocol type that issued the set mode request. It is passed as a longword value. On CSMA/CD, if padding is enabled on Ethernet format packets, the driver adds a 2-byte count field to the transmitted data. This field allows short packets (packets fewer than 46 bytes long) to be received with the proper length returned by the driver. The minimum Ethernet packet contains 46 bytes of user data. When fewer than 46 bytes are sent, the packet is padded and the receiver always receives 46 bytes of data. When padding is enabled, the maximum message size for transmit or receive operations is 1498 bytes and the minimum is zero bytes. See Section 8.6.1.2 for additional information. NMA$C_PCLI_PAD should be specified only on a port where the Ethernet packet format is selected (NMA$C_PCLI_FMT is set to NMA$C_LINFM_ETH). For FDDI, the same 2-byte count field is added. However, because FDDI packets can be as short as 22 bytes, FDDI transmit requests are never padded.

(continued on next page)

Table 8–11 (Cont.) P2 Attributes

Parameter ID	Meaning
NMA$C_PCLI_PHA[1]	Physical address (optional). It is passed as a counted string that consists of a modifier word followed by the 48-bit physical address. If the request is to clear the physical address or to set the physical address to the default address, the physical address (if present) is not read. One of the following mode values can be specified in the low byte of the modifier word: NMA$C_LINMC_SET — Set the string value. NMA$C_LINMC_CLR — Clear the physical address. NMA$C_LINMC_SDF — Set the physical address to the default address. For CSMA/CD, the default address is constructed by appending the low-order word of the SYSGEN parameter SCSSYSTEMID to the constant DECnet header (AA-00-04-00). If SCSSYSTEMID is zero, and NMA$C_LINMC_SDF is specified, the hardware address is used as the default. For FDDI, the default address is the hardware address. If not specified for CSMA/CD, the default is the current address set by a previous set mode function on this controller, or the hardware address if no address was defined by a previous set mode function. If not specified for FDDI, the default is the hardware address. The physical address must be passed as a 6-byte (48-bit) quantity. The first byte is the least significant byte. A return value of -1 on a sense mode request implies that a physical address is not defined. The NMA$C_PCLI_PHA parameter affects all ports on a single controller. If the address specified is already being used on the extended LAN, SS$_IVADDR is returned.
NMA$C_PCLI_PID	Protocol identifier. This parameter is required for, and valid only on, ports that use 802 extended format packets. NMA$C_PCLI_PID is passed as a counted 5-byte string, which is the unique protocol identifier required for each 802 extended format user. All protocol identifiers specified on a controller must be unique on that controller except when the PID is being shared.
NMA$C_PCLI_PRM	Promiscuous mode (optional). One of the following values can be specified: NMA$C_STATE_ON — Promiscuous mode enabled NMA$C_STATE_OFF — Promiscuous mode disabled (default) Only one port on each controller can be active with promiscuous mode enabled. Enabling promiscuous mode requires PHY_IO privilege. The NMA$C_PCLI_PRM parameter is passed as a longword value. Digital does not recommend promiscuous mode for normal usage. Some Token Ring devices do not support real promiscuous access to the ring. See Section 8.10.1 for additional information.

[1]If the LAN controller is active and you do not specify this parameter, the parameter defaults to the current setting. If the LAN controller is not active, this parameter defaults to the default value indicated.

(continued on next page)

Table 8–11 (Cont.) P2 Attributes

Parameter ID	Meaning
NMA$C_PCLI_PTY	Protocol type. This value is read as a 16-bit unsigned integer and must be different from other protocol types running on the same controller except when the protocol type is being shared. For Ethernet format ports, this is a required parameter. Valid protocol types are in the range 05-DD through FF-FF. NMA$C_PCLI_PTY should only be specified on a port where the Ethernet packet format is selected (NMA$C_PCLI_FMT is set to NMA$C_LINFM_ETH). NMA$C_PCLI_PTY is passed as a longword value; however, only the low-order word is used.
‡NMA$C_PCLI_RAC	Receive access control (Token Ring only). This optional parameter specifies whether the application receives a copy of the access control (AC) field for each Token Ring frame received. It is passed as a longword value. It must be passed with one of the following values: • NMA$C_STATE_ON — Application gets a copy of the AC for each Token Ring frame received. • NMA$C_STATE_OFF — Application does not get a copy of the AC for each Token Ring frame received. The AC is returned in the P5 buffer. The P5 buffer size for Token Ring should always be a minimum of 54 bytes. This is due to the variable size of the Token Ring header.
NMA$C_PCLI_RES	Restart. This optional parameter allows the user to enable the automatic port restart feature of the LAN drivers. One of the following values can be specified: NMA$C_LINRES_DIS — Disable automatic restart (default) NMA$C_LINRES_ENA — Enable automatic restart The LAN drivers shut down all users of a controller if there is a fatal error on the controller or if the LAN driver determines that the controller has stopped functioning. All outstanding I/O operations on the LAN driver are completed with either an SS$_ABORT or SS$_TIMEOUT status. All ports that have the NMA$C_PCLI_RES parameter enabled (set to NMA$C_ LINRES_ENA) have the port automatically restarted by the LAN driver approximately one second after it has been shut down due to a fatal error. If the user issues read or write QIOs to the port during the time the port is shut down, the driver completes the QIOs with an SS$_OPINCOMPL status. All ports that have the automatic restart feature disabled must be restarted by the application program when the port is shut down by the LAN driver. The application program should wait approximately 2 seconds to allow the LAN driver to stabilize. Once the LAN driver shuts down a port, it attempts a maximum of 30 consecutive automatic restarts. If there are 30 consecutive failures to restart the port, the port remains shut down. Note that it is unusual to have fatal errors on a LAN controller or to have a LAN driver detect that a LAN controller has stopped functioning. Having the ability to automatically restart a user's port makes the program easier to design because the program does not have to take into account the possibility of the LAN driver shutting down the port.

‡Alpha specific.

(continued on next page)

Table 8–11 (Cont.) P2 Attributes

Parameter ID	Meaning
NMA$C_PCLI_RFC	Receive frame control (FDDI only). This optional parameter specifies whether the application receives a copy of the Frame Control (FC) field for each FDDI frame received. It is passed as a longword value. However, only the low-order byte is used. It must be passed with one of the following values: • NMA$C_STATE_ON — Application gets a copy of the FC for each FDDI frame received. • NMA$C_STATE_OFF — Application does not get a copy of the FC for received FDDI frames (default). For $QIO Read operations, the FC is passed to the application in the P5 buffer. The following are the sizes required for the P5 buffer for various packet formats and settings of NMA$C_PCLI_RFC: • Ethernet (NMA$C_LINFM_ETH) — 14 if NMA$C_STATE_OFF is specified, 15 if NMA$C_STATE_ON is specified. • 802 (NMA$C_LINFM_802) — 16 if NMA$C_STATE_OFF is specified, 17 if NMA$C_STATE_ON is specified. • 802E (NMA$C_LINKFM_802E) — 20 if NMA$C_STATE_OFF is specified, 21 if NMA$C_STATE_ON is specified. Receiving the FC requires one additional byte of space in the P5 buffer. The FC is the first byte in the P5 buffer, immediately preceding the 6-byte destination address. The size of the P5 buffer required does not change from the CSMA/CD sizes if NMA$C_PCLI_RFC is set to NMA$C_STATE_OFF.
NMA$C_PCLI_SAP	802 format SAP. This parameter is required if the 802 packet format is selected (NMA$C_PCLI_FMT is set to NMA$C_LINFM_802). NMA$C_PCLI_SAP defines an 802 SAP and is read as an 8-bit unsigned integer. The least significant bit of the SAP must be zero and the SAP cannot be the null SAP (all 8 bits equal zero) or the SNAP SAP. NMA$C_PCLI_SAP is passed as a longword value. However, only the low-order byte is used. The SAP specified by NMA$C_PCLI_SAP is the SAP used to match incoming packets to complete read requests. It is used as the source SAP (SSAP) in all transmissions (write QIOs). Because it is illegal to transmit using a group SAP as the source SAP, the SAP specified by this NMA$C_PCLI_SAP cannot be a group SAP. NMA$C_PCLI_GSP describes how to set up group SAPs on a port. All individual SAPs specified on a controller must be unique on that controller. Therefore, the SAP specified using the NMA$C_PCLI_SAP parameter is checked for uniqueness on the controller.
‡NMA$C_PCLI_ SRMODE	Sets the source routing (SR) mode for the $QIO user (Token Ring only). This optional parameter allows the application to perform the source routing discovery. It must be passed with one of the following values: • NMA$C_SR_TRANSPARENT — Application source routing is transparent. This is the default when this parameter is not specifiied. • NMA$C_SR_SELF — This shuts off the automatic route discovery exploration messages for this user. The $QIOs exist to further manipulate the source routing cache. Digital recommends that applications use the NMA$C_SR_TRANSPARENT mode.

‡Alpha specific.

(continued on next page)

Table 8–11 (Cont.) P2 Attributes

Parameter ID	Meaning
NMA$C_PCLI_SRV	Port service. This optional parameter specifies the service supplied by the driver for the port. It can only be specified if the 802 packet format is selected (NMA$C_PCLI_FMT is set to NMA$C_LINFM_802). This characteristic is passed as a longword value. One of the following values can be specified: NMA$C_LINSR_USR — User-supplied service (default) NMA$C_LINSR_CLI — Class I service See Section 8.6.2.1 for a description of Class I service and Section 8.6.2.2 for a description of user-supplied service.
‡NMC$C_PCLI_XAC	Transmit access control (Token Ring only). This is an optional parameter that enables applications to control the setting of the priority bits in the access control (AC) for frames being transmitted in a $QIO write operation. When set to the wamted value, all subsequent transmits use this AC value.
NMA$C_PCLI_XFC	Transmit frame control (FDDI only). NMA$C_PCLI_XFC is an optional parameter that enables applications to control the setting of the priority bits in the FC for frames being transmitted in a $QIO write operation. NMA$C_ PCLI_XFC is passed as a longword parameter that has many valid settings. If specified with a value of zero, the application supplies an FC value on each $QIO write operation. The FC value to be used in this case is supplied in the P5 buffer for the $QIO write operation. If the parameter is specified with a value other than zero, that value is inserted into the FC field of every transmit by the FDDI drivers. No FC is present in the P5 buffer for the $QIO write in this case. If this parameter is not specified, the default setting (zero) of the priority bits is used. Regardless of how the FC is supplied, the value specified must be valid. The allowable values for FC are between FC_LLC_MIN and FC_LLC_MAX. If NMA$C_PCLI_XFC is specified with a nonzero value outside the valid range, the application receives a SS$_BADPARAM error. FC_LLC_MIN is 80 (50 hex) because the priority bits are the three low-order bits, FC_LLC_MAX is 87 (57 hex). An application that needs to support large buffers must use a value for NMA$C_PCLI_XFC that is at least one greater than FC_LLC_MIN because a value less then FC_LLC_MIN would not correctly indicate that large buffers are supported.

‡Alpha specific.

8.8.3.2 Set Mode Parameters for Packet Formats

Table 8–12 summarizes the use of the set mode parameters for the Ethernet, 802, and 802 extended (802E) packet formats.

Table 8–12 Set Mode Parameters for Packet Formats

Parameter ID	Ethernet	IEEE 802	802E
FMT	DEF	REQ	REQ
PTY	REQ	E	E

Legend

DEF—Default. If not specified, this is the default parameter for this packet format.
REQ—Required. This parameter must be specified for this packet format.
OPT—Optional. This parameter is optional for this packet format; it may be specified.
E—Error. This parameter cannot be specified for this packet format. If the parameter is specified, it generates an SS$_BADPARAM error.

(continued on next page)

Table 8–12 (Cont.) Set Mode Parameters for Packet Formats

Parameter ID	Ethernet	IEEE 802	802E
SAP	E	REQ	E
PID	E	E	REQ
ACC	OPT	E	†E
DES	OPT	E	†E
PAD	OPT	E	E
SRV	E	OPT	E
GSP	E	OPT	E
BFN, BUS, CCA, CON, CRC, EKO, ILP, MCA, MLT, PHA, PRM, ‡RAC, RES, RFC, ‡SRMODE, ‡XAC, XFC	OPT	OPT	OPT

†VAX specific. On Alpha systems, however, the ACC and DES parameters are OPT.

‡Alpha specific.

Legend

DEF—Default. If not specified, this is the default parameter for this packet format.
REQ—Required. This parameter must be specified for this packet format.
OPT—Optional. This parameter is optional for this packet format; it may be specified.
E—Error. This parameter cannot be specified for this packet format. If the parameter is specified, it generates an SS$_BADPARAM error.

8.8.3.3 Set Mode Parameter Validation

When starting a LAN port, the LAN driver checks that the mode of the new port is compatible with the mode of the LAN ports already started. There are two sets of compatibility checks: one for ports running in shared mode and one for all ports.

The following parameters must match for all ports on the same controller:

NMA$C_PCLI_CON
NMA$C_PCLI_CRC
NMA$C_PCLI_EKO
NMA$C_PCLI_ILP
NMA$C_PCLI_PHA (need only match for CSMA/CD controllers)

VAX

The following parameters must match for all shared-default and shared-with-destination users of the same protocol type:

NMA$C_PCLI_BFN
NMA$C_PCLI_BUS
NMA$C_PCLI_CCA
NMA$C_PCLI_MLT
NMA$C_PCLI_PAD
NMA$C_PCLI_PTY
NMA$C_PCLI_RAC
NMA$C_PCLI_RES
NMA$C_PCLI_RFC
NMA$C_PCLI_XAC
NMA$C_PCLI_XFC ♦

Once a port is started, only the following parameters can be changed:

NMA$C_PCLI_GSP
NMA$C_PCLI_MCA

8.8.3.4 Shutdown Controller

The shutdown controller function shuts down the LAN port. On completion of a shutdown request all outstanding I/O requests are completed. This port cannot be used again until another startup request has been issued (see Section 8.8.3.1).

The following function code is used to shut down a port:

IO$_SETMODE!IO$M_CTRL!IO$M_SHUTDOWN—Shut down port

The shutdown controller function takes no device- or function-dependent arguments.

8.8.3.5 Enable Attention AST

This function requests that an attention AST be delivered to the requesting process when a status change occurs on the assigned port. An AST is queued when a message is available and there is no waiting read request. The enable attention AST function is legal at any time, regardless of the condition of the unit status bits.

The following function code and modifier is used to enable an attention AST:

- IO$_SETMODE!IO$M_ATTNAST—Enable attention AST

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

- P1—The address of an AST service routine or 0 for disable
- P2—Ignored
- P3—Access mode to deliver AST

The enable attention AST function enables an attention AST to be delivered to the requesting process once only. After the AST occurs, it must be explicitly reenabled by the function before the AST can occur again. The function is subject to AST quotas.

The AST service routine is called with an argument list. The first argument is the current value of the second longword of the I/O status block (see Section 8.9).

8.8.3.6 IO$M_SET_MAC Functional Modifier to IO$M_SETMODE

The IO$M_SET_MAC qualifier, when used with IO$_SETMODE, is used to set medium specific parmeters. The Token Ring parameters require PHY_IO privilege to set. Table 8–13 shows the parameters that may be set.

Table 8–13 Medium Specific Parameters of IO$M_SET_MAC

Parameter ID	Meaning
NMA$C_PCLI_TREQ	Requested value for token rotation timer, ANSI MAC T_req parameter (FDDI only). Units are in 80 nanoseconds, default is 8000, minimum is 4000, and maximum is 167772.
NMA$C_PCLI_TVX	Maximum time between arrivals of a valid frame or unrestricted token, ANSI MAC TVX parameter (FDDI only). Units are in 80 nanoseconds, default is 2621, minimum is 2500, and maximum is 5222.
NMA$C_PCLI_REST_TTO	Restricted token timeout which limits how long a single restricted mode dialog may last before being terminated (FDDI only). Units are in milliseconds, default is 1000, minimum is 0, and maximum is 10000.
NMA$C_PCLI_RPE	Ring purge enable. If 1 (TRUE), this link will particpate in the Ring Purger election, and, if elected, perform the Ring Purger function (FDDI only).
NMA$C_PCLI_NIF_TARG	Neighbor information frame target (FDDI only).
NMA$C_PCLI_SIF_CONF_ TARG	Station information frame configuration target (FDDI only). A 6-byte string specifying the LAN address of the target. Used only by DECnet/OSI.
NMA$C_PCLI_SIF_OP_TARG	Station information frame operation target (FDDI only). A 6-byte string specifying the LAN address of the target. Used only by DECnet/OSI.
NMA$C_PCLI_ECHO_TARG	Echo test target (FDDI only). A 6-byte string specifying the LAN address of the target. Used only by DECnet/OSI.
NMA$C_PCLI_ECHO_DAT	Data pattern to use for the echo test (FDDI only). Used only by DECnet/OSI.
NMA$C_PCLI_ECHO_LEN	Length of the echo packet (FDDI only). Used only by DECnet/OSI.
‡NMA$C_PCLI_RNG_SPD	Sets the speed of the ring (Token Ring only). This longword may be either: NMA$C_LINRNG_FOUR—Used for 4 Mbps rings. NMA$C_LINRNG_SIXTN—Used for 16 Mbps rings. The default is NMA$C_LINRNG_SIXTN.

‡Alpha specific.

(continued on next page)

Table 8–13 (Cont.) Medium Specific Parameters of IO$M_SET_MAC

Parameter ID	Meaning
‡NMA$C_PCLI_LINEMEDIA	Sets the connection media type for the Token Ring adapter (Token Ring only). Valid values for this longword parameter are: NMA$C_MEDIA_STP NMA$C_MEDIA_UTP The default is NMA$C_MEDIA_STP.
‡NMA$C_PCLI_ETR	Controls the Early Token release feature of the Token Ring hardware (Token Ring only). This feature can greatly improve throughput, and is only valid on 16 Mb/s rings. The values for this longword parameter are NMA$C_STATE_ON or NMA$C_STATE_OFF. The default is NMA$C_STATE_ON.
‡NMA$C_PCLI_MONCONTEND	Specifies whether the controller participates in the monitor contention process when another adapter detects the need for contention and initiates the process (Token Ring only). The values for this longword parameter are NMA$C_STATE_ON or NMA$C_STATE_OFF. The default is NMA$C_STATE_OFF.
‡NMA$C_PCLI_CACHE_ENT	The number of source routing (SR) entries to make available for caching (Token Ring only). The default is 200, minimum is 20, and maximum is 2000. Each cache entry consumes 64 bytes.
‡NMA$C_PCLI_ROUTEDIS	The source routing discovery timer (Token Ring only). This is the amount of seconds to wait after the transmission of ring explorer packets before declaring the route of a path to be unknown. The default is 2 seconds, minimum is 1, and maximum is 255.
‡NMA$C_PCLI_A_TIM	The source routing aging timer (Token Ring only). After traffic is neither received from nor sent to a given node for this number of seconds, the entry is marked stale. After the entry is marked stale, rediscovery is required to communicate with the node. The default is 60 seconds, minium is 1, and maximum is 65535.
‡NMA$C_PCLI_SRC_ROU	Enables and disables source routing (Token Ring only). The values for this longword parameter are NMA$C_LINSRC_ENA or NMA$C_LINSRC_DIS. The default is NMA$C_LINSRC_ENA.
‡NMA$C_PCLI_AUTH_PR	Specifies the highest priority that a user may transmit a frame (Token Ring only). The priority is set within the NMA$C_PCLI_XAC parameter. The default for this parameter is 3, minimum is 0, and maximum is 6.

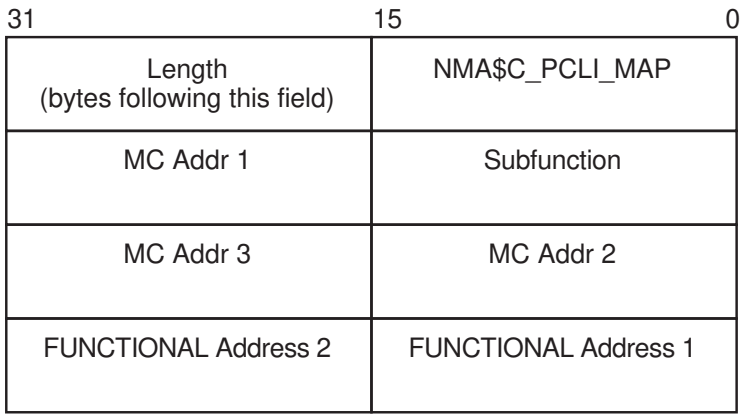
‡Alpha specific.

8.8.3.7 IO$M_UPDATE_MAP Functional Modifier to IO$_SETMODE (Alpha Only)

Alpha

On Alpha systems using Token Ring only, the IO$M_UPDATE_MAP qualifier, when used with IO$_SETMODE, manipulates the adapter's functional address mapping table. Figure 8–22 shows the format of the P2 buffer for this operation. This QIO requires PHY_IO privilege.

Figure 8–22 Format of IO$M_UPDATE_MAP Setmode P2 Buffer (Alpha Only)



ZK–6791A–GE

The subfunction is one of the following:

- NMA$C_MAP_CHANGE—This function adds or changes a mapping in the functional address table. If the specified multicast entry does not exist, an entry is created with the specified functional address mask. If the specified multicast entry does exist, the corresponding functional address mask is changed to the specified mask. All users who currently have the multicast enabled when the functional mask is changed will automatically update the functional address table as part of this operation.

  Possible errors returned include the following:

  SS$_DEVICEFULL—This error indicates that there is insufficient space in the mapping table to complete the request. The multicast to functional address mapping table has 200 entries.

- NMA$C_MAP_DELETE—This function deletes the specified MC address in the table. For this function, the functional address mask is not required to pass the P2 buffer. If the functional address mask is passed, its contents are ignored.

  Possible errors returned includes the following:

  SS$_BADPARAM — This error indicates that the specified multicast address cannot be found in the table.

The following example maps multicast address AB-01-01-01-02-03 to the functional address 03-00-00-01-00-00 for device ICA0:.

```
LANCP>SET DEVICE/MAP= -
_LANCP>(MULTICAST=AB-01-01-01-02-03, -
_LANCP>FUNCTIONAL=00-01-00-00) ICA0:
```

The following example deletes the mapping of the multicast address of AB-01-01-01-02-03 for the device ICA0:.

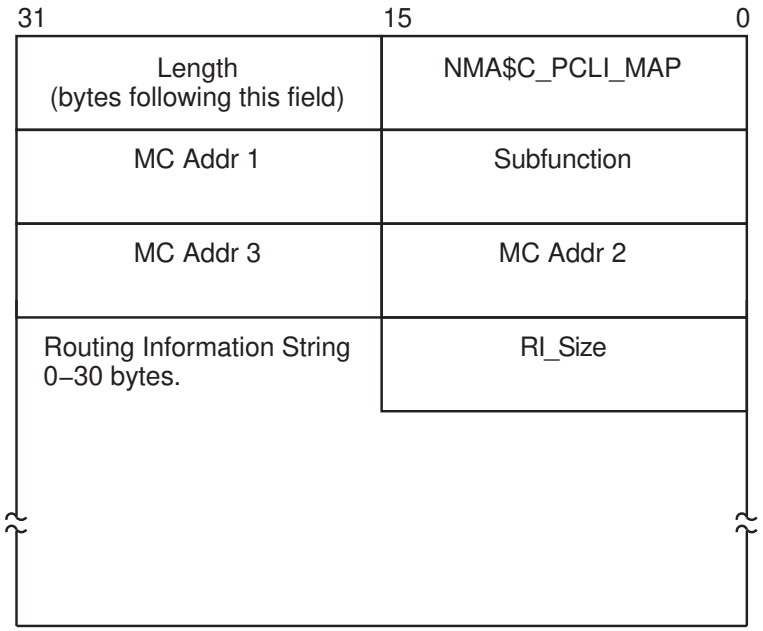
```
LANCP>SET DEVICE/NOMAP=(MULTICAST=AB-01-01-01-02-03) ICA0:♦
```

8.8.3.8 IO$M_ROUTE Functional Modifier to IO$_SETMODE (Alpha Only)

Alpha

On Alpha systems using Token Ring only, the IO$M_ROUTE qualifier, when used with IO$_SETMODE, manipulates the source routing cache table. This command is successful only when source routing is enabled. Source routing is enabled with the set mac qualified set mode QIO. Figure 8–23 shows the format of the P2 buffer. This QIO require PHY_IO privilege.

Figure 8–23 Format of the IO$M_ROUTE P2 Buffer (Alpha Only)



The subfunction is one of the following:

- NMA$C_SR_ADD—This function adds or changes a source routing cache entry. It enters the LAN address into the table with the enclosed routing information. The routing information string format is documented in Section 8.4.5. If RI_size is passed as 0, the entry is created (or modified) to be in the EXPLORING state (this is useful for users who are doing their own source routing). If the RC 'Lth' field is zero, the LAN address is entered in the table as being in the local state.

  Possible errors returned include:

  SS$_INSFMEM—The source routing cache is full.
  SS$_BADPARAM—An invalid RI string was passed or invalid sizes were passed.
  SS$_IVMODE—Source routing is not enabled.

- NMA$C_SR_DELETE—This function deletes a source routing cache entry. The RI_size and the routing information string are not required for this QIO. If one or both of the fields are passed for this operation, they are ignored. The result of this command is to put the entry into the deleted state. When the entry goes into the deleted state, it is deleted within 10 minutes.

Possible errors returned includes the following:

SS$_BADPARAM—The requested entry could not be found. ♦

8.8.4 Sense Mode and Sense Characteristics

The sense mode function returns the port attributes in the specified buffers. These attributes include the device characteristics described in Section 8.7 and, with the exceptions noted below, the attributes listed in Table 8–11.

The following combinations of function code and modifier are provided:

- IO$_SENSEMODE!IO$M_CTRL—Read characteristics
- IO$_SENSECHAR!IO$M_CTRL—Read characteristics
- IO$_SENSEMODE!IO$M_SENSE_MAC—Medium specific characteristics

Alpha

The following combinations of function code and modifier for Token Ring are provided:

- IO$_SENSEMODE!IO$M_SHOW_MAP—Returns current functional address to multicast address mapping
- IO$_SENSEMODE!IO$M_SHOW_ROUTE—Returns current source routing cache table ♦

These functions take the following device- or function-dependent arguments:

- P1—The address of a two-longword buffer where the device characteristics are stored. (Figure 8–24 shows the format for, and Section 8.7 describes the contents of, the P1 buffer.) The P1 argument is optional.
- P2—The address of a quadword descriptor where the attributes buffer is stored. The first longword of the descriptor is the buffer length; the second longword is the address of the buffer. The P2 argument is optional.

  The P2 buffer is not read by the LAN driver. The driver stores the port's attributes in the buffer, which contains multiple entries. The format of each entry depends on whether a longword or a counted string is returned, as shown in Figure 8–25. Each parameter ID contains a string indicator bit (bit 12) that describes whether the data item is a string or a longword.

Except for the following differences, P2 returns the same attributes as those listed in Table 8–11:

- All parameters that are valid for the enabled packet format are returned (see Table 8–12).
- The sense-mode P2 buffer does not return the modifier word for the NMAC_PCLI_PHA, NMAC_PCLI_MCA, and NMA$C_PCLI_DES parameter IDs.
- The NMA$C_PCLI_DES parameter is only returned on Ethernet ports whose access mode is set to ''shared with destination.''
- In addition to the parameter IDs listed in Table 8–11, the sense-mode P2 buffer contains the following parameter IDs:

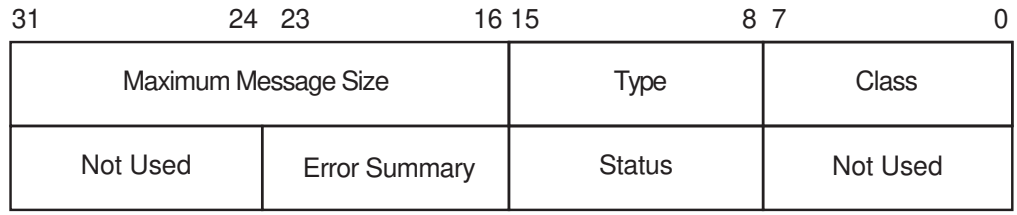
Parameter ID	Meaning
‡NMA$C_PCLI_FCA	List of the currently enabled functional addresses (Token Ring only). Each 32-bit entry corresponds respectively with the items returned under NMA$C_PCLI_MCA.
NMA$C_PCLI_HWA	Hardware address. Describes the value for the hardware address. The hardware address is the default physical address when no physical address has been specified and there are no active users on the controller. NMA$C_PCLI_ HWA is returned in the same format as NMA$C_PCLI_PHA.
NMA$C_PCLI_MBS	Maximum packet length. NMA$C_PCLI_MBS is a longword, read-only parameter. The value returned reflects the largest data packet that the application can receive for its packet format and type of LAN, measured in bytes. The values are:

Packet format	CSMA/CD	FDDI	‡Token Ring
Ethernet	1500	4470	‡4418
Ethernet with PAD	1498	4468	‡4416
802	1497	4475	‡4423
802E	1492	4470	‡4418

NMA$C_PCLI_MED	Type of medium to which the device is attached. Current media supported are NMA$C_LINMD_FDDI (FDDI), NMA$C_LINMD_CSMACD (Ethernet and IEEE 802.3), and NMA$C_LINMD_TR (Token Ring). This is a longword parameter.

‡Alpha specific.

Figure 8–24 Sense Mode P1 Characteristics Buffer

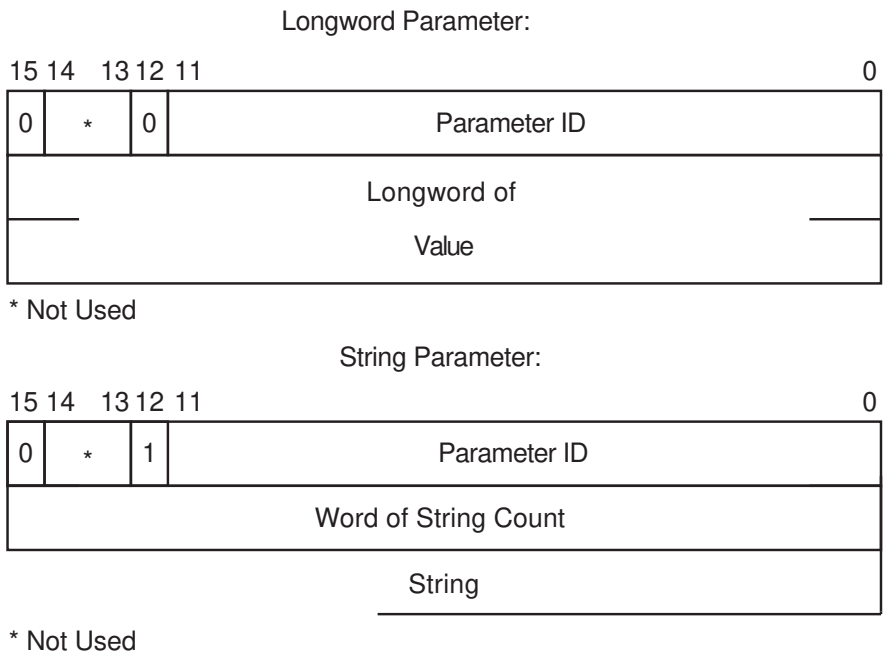


ZK–1178–GE

It is suggested that a size of 250 bytes be used for the P2 buffer. This will allow space for additional parameters that may be returned in future releases of OpenVMS.

All attributes that fit into the buffer specified by P2 are returned. However, if all the attributes cannot be stored in the buffer, the I/O status block returns the status SS$_BUFFEROVF. The second word of the I/O status block contains the number of bytes used in the P2 buffer (see Section 8.9).

Figure 8–25 Sense Mode Attribute Buffer



8.8.4.1 IO$M_SENSE_MAC Functional Modifier to IO$_SENSEMODE

The IO$M_SENSE_MAC qualifier, when used with IO$_SENSEMODE, returns the parameters specified in Section 8.8.3.6. In addition to the set mac parameters, Table 8–14 shows the returns of the following parameters:

Table 8–14 Parameters of IO$M_SENSE_MAC

Parameter ID	Meaning
NMA$C_PCLI_T_NEG	The negotiated value of the token rotation timer (ANSI MAC parameter T_neg) (FDDI only).
NMA$C_PCLI_DAT	The duplicate address test flag (FDDI only). If set, this indicates that there is another station on the ring with the same hardware LAN address.
‡NMA$C_PCLI_UNA	Upstream neighbor's address (FDDI and Token Ring). This is a string parameter specifying the 6-byte LAN address of the upstream neighbor. Not all devices may support this feature.
NMA$C_PCLI_OLD_ UNA	The old (previous) upstream neighbor address (FDDI only). Neighbor addresses change as nodes insert and deinsert into the ring.
NMA$C_PCLI_UN_DAT	The upstream neighbor's duplicate address test flag (FDDI only).
NMA$C_PCLI_DNA	The downstream neighbor's LAN address (FDDI only).
NMA$C_PCLI_OLD_ DNA	The old (previous) downstream neighbor's LAN address (FDDI only).

‡Alpha specific.

(continued on next page)

Table 8–14 (Cont.) Parameters of IO$M_SENSE_MAC

Parameter ID	Meaning
NMA$C_PCLI_RPS	The current ring purger state (FDDI only). This longword parameter is one of the following values: 0—Off 1—Candidate 2—Non-purger 3—Purger
NMA$C_PCLI_RER	The latest ring error reason (FDDI only). This longword parameter is one of the following values: 0—No Error 5—Ring Init initiated 6—Ring Init received 7—Ring beaconing initiated 8—Duplicate address detected 9—Duplicate token detected 10—Ring purger error 11—FCI strip error 12—Ring op oscillation 14—PC trace initiated 15—PC trace received
NMA$C_PCLI_NBR_PHY	Neighbor's PHY type (FDDI only). This longword parameter is one of the following values: 0—A 1—B 2—S 3—M 4—Unknown
NMA$C_PCLI_RJR	Ring reject reason (FDDI only). This longword parameter is one of the following values: 0—None 1—Local LCT 2—Remote LCT 3—LCT both sides 4—LEM reject 5—Topology error 6—Noise reject 7—Remote reject 8—Trace in progress 9—Trace received-disabled 10—Standby 11—LCT protocol error
NMA$C_PCLI_LEE	Link error estimate (FDDI only). The longword value is a negative exponent of 10 representing the Link error rate. For example, the value of X represents the error rate of 10^X.
‡NMA$C_PCLI_RNG_NUM	The longword value contains the ring number that the controller is running on (Token Ring only). It is only valid for a controller that is started, and also only valid for rings that have a ring parameter server that is configured for providing this information.

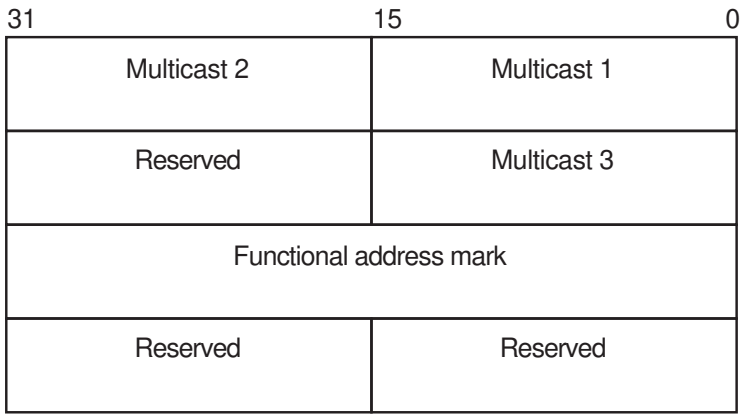
‡Alpha specific.

8.8.4.2 IO$M_SHOW_MAP Functional Modifier to IO$_SENSMODE (Alpha Only)

Alpha

On Alpha systems using Token Ring only, the IO$M_SHOW_MAP qualifier, when used with IO$_SENSEMODE, returns the current setting of the mapping table. The P2 buffer is filled with the current multicast to functional address mapping information. The entries are 16 bytes long and are in the format shown in Figure 8–26. This QIO requires PHY_IO privilege.

Figure 8–26 Format of IO$M_SHOW_MAP P2 Buffer (Alpha Only)



The multicast address and functional address mask are returned in canonical format (that is, not bit-reversed). The following errors may occur:

- SS$_BUFFEROVF—The passed buffer is not large enough to hold all the data required for the operation.
- SS$_BADPARAM—Not able to get read access to buffer or zero length buffer passed. ♦

8.8.4.3 IO$M_SHOW_ROUTE Functional Modifier to IO$_SENSEMODE (Alpha Only)

Alpha

On Alpha systems with Token Ring only, the IO$M_SHOW_ROUTE qualifier, when used with IO$_SENSEMODE, returns the current value of the source routing cache table. Each entry is 64 bytes long. Figure 8–27 shows the format of the returned P2 buffer:

Figure 8–27 Format of IO$M_SHOW_ROUTE P2 Buffer (Alpha Only)

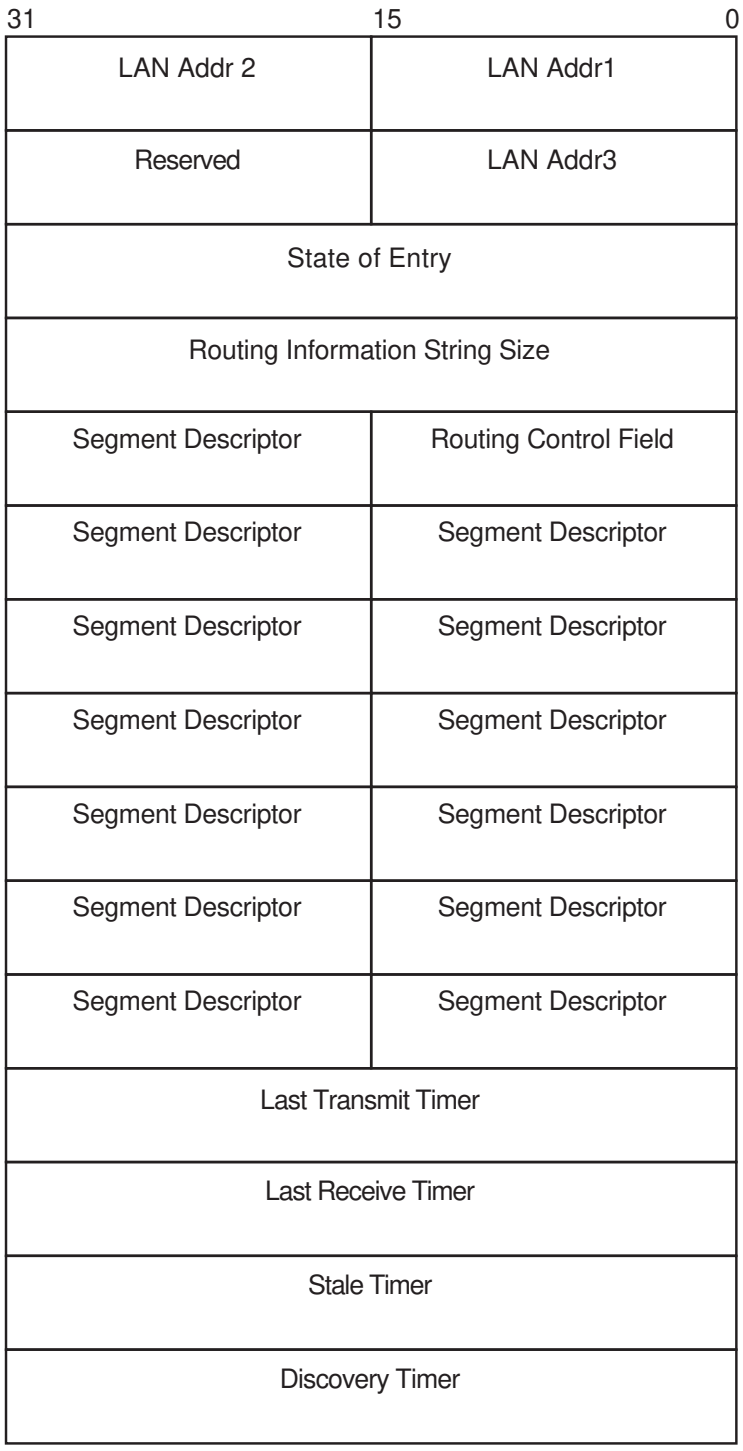


Table 8–15 shows possible states of the entry.

Table 8–15 State of the Entry

Value	Name	Description
0	LOCAL	Address is reachable on the attached ring.
1	STALE	Entry is stale (inactive).
2	UNKNOWN	Route to the address is unknown.
3	DELETED	Entry is marked for deletion.
4	KNOWN	Route is known and the route is stored in the routing information string.
5	EXPLORING	Route to the address is currently being explored.

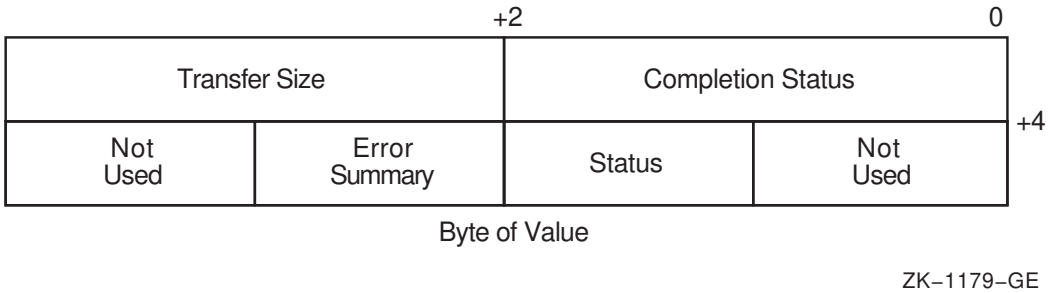
The LAN address is returned in canonical format (that is, not bit-reversed). The timers are recorded as seconds before expiration. The transmit and receive timers are initialized from the NMA$C_PCLI_A_TIM parameter, the discovery timer is initialized from the NMA$C_PCLI_ROUTEDIS parameter, and the stale timer is initialized to 10 minutes (600 seconds). The following errors may occur:

- SS$_BUFFEROVF—The passed buffer is not large enough to hold all the data required for the operation.
- SS$_BADPARAM—Not able to get read access to buffer or zero length buffer passed. ♦

8.9 I/O Status Block

The I/O status block (IOSB) for all LAN driver functions is shown in Figure 8–28. Appendix A lists the completion status returns for these functions. (The OpenVMS system messages documentation provides explanations and suggested user actions for these returns.)

Figure 8–28 IOSB Contents



The first longword of the IOSB returns, in addition to the completion status, either the size (in bytes) of the data transfer or the size (in bytes) of the attribute buffer (P2) returned by a sense mode function. The second longword returns the unit and line status bits listed in Table 8–6 and the error summary bits listed in Table 8–7.

8.10 Application Programming Notes

This section contains information to assist you in writing application programs that use the LAN device drivers. Section 8.10.1 discusses the additional rules required for application programs that you intend to run in promiscuous mode. Section 8.10.2 describe the Ethernet and 802 sample programs.

8.10.1 Promiscuous Mode

The LAN drivers allow only one port per controller to start with promiscuous mode enabled (NMA$C_PCLI_PRM specified as NMA$C_STATE_ON). Any port running in promiscuous mode usually places an additional load on the CPU because the LAN driver processes every packet on the LAN for the promiscuous user. If there is no promiscuous port on a controller, the controller performs most of the filtering required for the packets on the line.

Table 8–16 details additional rules for ports running in promiscuous mode.

Table 8–16 Rules for Promiscuous Mode Operation

I/O Function	Rule
IO$_SETMODE IO$_SETCHAR	It is not necessary to specify a unique identifier (a protocol type, SAP, or protocol identifier parameter ID) in the P2 buffer. The port cannot be running in shared mode.
IO$_WRITE	The user can only transmit packets in the packet format previously enabled with a set mode QIO. The unique identifier for the packet format must be included in the P5 buffer following the destination address (see Section 8.8.2).
IO$_READ	The LAN driver completes the promiscuous user's read requests with Ethernet, IEEE 802, and 802 extended packets. Because any packet format can be used to complete a read request, the P5 parameter (if specified) must be 20 bytes in length (21 bytes for FDDI with RFC turned on). All Ethernet format packets are processed as if they have no size word specified after the protocol type. Therefore, Ethernet packets are always returned with 46 to 1500 bytes of data. If the Ethernet packet contains a size word, it is returned as part of the user data in the first word of the P1 buffer. The promiscuous user should use the information returned in the P5 buffer to determine the packet format. If the application program first filled the P5 buffer with zeros, the program should be able to determine the format of the packet received by scanning the P5 buffer after a read request is completed.

8.10.2 Local Area Network Programming Examples

The VAX MACRO program LANETH.MAR (Example 8–5) shows the typical use of QIO functions in driver operations such as establishing the protocol type, starting the port, and transmitting and receiving data. The program sends a LOOPBACK packet and waits for the packet to be returned.

The VAX MACRO program LAN802.MAR (Example 8–6) shows how to initialize an IEEE 802 port and how to send and receive packets on that port. This program sends a TEST packet and waits for the TEST response.

Example 8–5 LANETH.MAR Local Area Network Programming Example

```
; ****************************************************************
;
        .TITLE  EXAMPLE           LAN SAMPLE TEST PROGRAM
        .IDENT  /X01/

; This LAN test program will send a LOOPBACK message to another system
; and wait for a response.  Since LOOPBACK forwarding is handled by the
; controller or the driver at the other node, you should always get a
; response as long as the other node exists.
;
; Note that this test will try to use the device defined by the logical
; LAN as the LAN device.  If this does not work, then it will try to use
; one of the currently known LAN devices.  To use a particular Ethernet
; or FDDI device, you need to define the logical LAN to be the name of the
; device you wish t use.  For example:
;
;       $ DEFINE LAN EXB0
;
; Note that if you have service enabled on a DECnet circuit enabled on the
; controller you wish to test, this program will get a fatal error
; when trying to start its port.  This is expected because DECnet will
; start its own port for the LOOPBACK protocol.

        .LIBRARY        "SYS$LIBRARY:LIB.MLB"

        $IODEF                          ; Define I/O functions and modifiers
        $NMADEF                         ; Define Network Management parameters

; Local definitions

        RCVBUFLEN = 512                 ; Size of receive buffer
        XMTBUFLEN = 20                  ; Size of transmit buffer

; Setmode parameter buffer.  You are required to state only the
; unique protocol type value.  However, you will also state the packet format.
; Since the LOOPBACK protocol does not include a LENGTH word following the
; protocol type, you have to explicitly turn OFF padding since the default is
; ON.


SETPARM:
        .WORD   NMA$C_PCLI_FMT          ; Packet format
                .LONG   NMA$C_LINFM_ETH
        .WORD   NMA$C_PCLI_PTY          ; Our Protocol type
                .LONG   ^X0090
        .WORD   NMA$C_PCLI_PAD          ; Padding
                .LONG   NMA$C_STATE_OFF

        SETPARMLEN = .-SETPARM

SETPARMDSC:
        .LONG           SETPARMLEN
        .ADDRESS        SETPARM

; Sensemode parameter buffer.  This will be used to get our node's physical
; address to put into the loopback message.

SENSEBUF:
        .BLKB   250
```

(continued on next page)

Example 8–5 (Cont.) LANETH.MAR Local Area Network Programming Example

```
SENSELEN=.-SENSEBUF
SENSEDSC:
        .LONG           SENSELEN
        .ADDRESS        SENSEBUF

; P2 transmit data buffer

XMTBUF:
        .WORD   00                      ; Skip count
        .WORD   02                      ; Forward request
FORW:   .BLKB   6                       ; You will put our address here
        .WORD   01                      ; Reply request
        .WORD   00

XMTBUFLEN = .-XMTBUF

; P5 transmit destination address
;
; Set this value to be a node on your LAN that supports LOOPBACK.

XMTP5:
        .BYTE   ^XAA,^X00,^X04,^X00,^X17,^X4E

; P2 receive data buffer

RCVBUF:
        .BLKB   RCVBUFLEN

; P5 receive header buffer

RCVP5:
RCVDA:  .BLKB   6
RCVSA:  .BLKB   6
RCVPTY: .BLKB   2

; Messages used to display status of this program.

GMSG:   .ASCID  "Successful test"
BMSG:   .ASCID  "Received packet was not what was expected"
LMSG:   .ASCID  "Packet lost or node not responding"
EMSG:   .ASCID  "Error occurred while running test"
DMSG:   .ASCID  "No LAN device found - please define LAN correctly"

; Miscellaneous data structures
```

(continued on next page)

Example 8–5 (Cont.) LANETH.MAR Local Area Network Programming Example

```
TRY:    .WORD   0                       ; Number of times you have tried
                                        ; the READ QIO (start at 0)
IOSB:   .BLKQ   1                       ; I/O status block
LANDSC1:.ASCID  'LAN'                   ; Units to use for test
LANDSC2:.ASCID  'ECA0'
LANDSC3:.ASCID  'ESA0'
LANDSC4:.ASCID  'ETA0'
LANDSC5:.ASCID  'EXA0'
LANDSC6:.ASCID  'EZA0'
LANDSC7:.ASCID  'FCA0'
LANDSC8:.ASCID  'FXA0'
LANDSC9:.ASCID  'ICA0
LANDSCA:.ASCID  'IRA0
LANDSC9:.ASCID  'XEA0'
LANDSCA:.ASCID  'XQA0'
DSCADR: .ADDRESS        LANDSC1         ; Table of addresses pointing to
        .ADDRESS        LANDSC2         ; the descriptors of device names
        .ADDRESS        LANDSC3
        .ADDRESS        LANDSC4
        .ADDRESS        LANDSC5
        .ADDRESS        LANDSC6
        .ADDRESS        LANDSC7
        .ADDRESS        LANDSC8
        .ADDRESS        LANDSC9
        .ADDRESS        LANDSCA
        .ADDRESS        LANDSCB
        .ADDRESS        LANDSCC
        .LONG           0
DEVCHAN:.BLKL   1                       ; Returned port number

;*****************************************************************************
;
; Start of code
;
;*****************************************************************************

        .ENTRY  START,^M<>

; Assign a port to the LAN device.  If LAN does not work, try each
; of the currently known LAN devices.

ASSIGN:
        MOVAL   DSCADR,R5               ; Start at beginning
10$:    MOVL    (R5),R4                 ; Is there a descriptor?
        BEQL    20$                     ; If EQL, no
        $ASSIGN_S DEVNAM=(R4),CHAN=DEVCHAN
        BLBS    R0,ASSIGN_OK
        ADDL    #4,R5                   ; Skip to next descriptor
        CMPW    R0,#SS$_NOSUCHDEV       ; Was the device not there?
        BEQL    10$                     ; If EQL, yes, try next device
        BRW     ERROR

; You could not find a LAN device to assign a port to.

20$:    PUSHAB  DMSG
        BRW     EXIT

ASSIGN_OK:

; Set up the port's characteristics.
```

(continued on next page)

Example 8–5 (Cont.) LANETH.MAR Local Area Network Programming Example

```
        $QIOW_S FUNC=#<IO$_SETMODE!IO$M_CTRL!IO$M_STARTUP>,-
                CHAN=DEVCHAN,IOSB=IOSB,-
                P2=#SETPARMDSC
        BLBS    R0,STARTUP_REQ_OK
        BRW     ERROR

STARTUP_REQ_OK:

        MOVZWL  IOSB,R0
        BLBS    R0,STARTUP_IO_OK
        BRW     ERROR


STARTUP_IO_OK:

; Now issue the SENSEMODE QIO so that you can get our physical address and
; put it in the LOOPBACK message you are about to transmit.

        $QIOW_S FUNC=#<IO$_SENSEMODE!IO$M_CTRL>,-
                CHAN=DEVCHAN,IOSB=IOSB,-
                P2=#SENSEDSC
        BLBS    R0,SENSE_REQ_OK
        BRW     ERROR

SENSE_REQ_OK:

        MOVZWL  IOSB,R0
        BLBS    R0,SENSE_IO_OK
        BRW     ERROR

SENSE_IO_OK:

; Now you have to locate the PHA parameter in the SENSEMODE buffer and copy
; it into our LOOPBACK transmit message.  You will scan the return buffer
; for a string parameter.  If you find a string parameter, you will check if
; it's the PHA parameter.

        MOVAB   SENSEBUF,R0             ; Start at beginning of buffer

10$:    BBS     #^XC,(R0),20$           ; If this is a string parameter,
                                        ; goto 20$

; Skip over the longword parameter.

        ADDL    #6,R0                   ; Skip 2-byte type and 4-byte value
        BRB     10$                     ; Check next parameter

; This is a string parameter.  Check if it's the PHA parameter.

20$:    BICW    #^XF000,(R0)            ; Clear flag bits in type field
        CMPW    #NMA$C_PCLI_PHA,(R0)    ; Is this the PHA parameter?
        BEQL    30$                     ; If EQL, yes

; Skip over this string parameter.

        ADDL    #2,R0                   ; Skip 2-byte type
        MOVZWL  (R0)+,R1                ; Convert string size to longword
                                        ; and skip it
        ADDL    R1,R0                   ; Skip string
        BRB     10$                     ; Check next parameter
```

(continued on next page)

Example 8–5 (Cont.) LANETH.MAR Local Area Network Programming Example

```
; You have located the PHA parameter.  Move it into the LOOPBACK transmit
; buffer.

30$:    MOVL    4(R0),FORW              ; Move 1st four bytes
        MOVW    8(R0),FORW+4            ; Move last two bytes

; Now transmit our TEST message.

        $QIOW_S FUNC=#IO$_WRITEVBLK,CHAN=DEVCHAN,IOSB=IOSB,-
                P1=XMTBUF,P2=#XMTBUFLEN,P5=#XMTP5
        BLBS    R0,XMIT_REQ_OK
        BRW     ERROR

XMIT_REQ_OK:

        MOVZWL  IOSB,R0
        BLBS    R0,XMIT_IO_OK
        BRW     ERROR

XMIT_IO_OK:

; Now try to receive the response.  You will use the NOW function modifier
; on the READ so that you don't hang here waiting forever if there is no
; response.  You will attempt to receive the message 1000 times.  If there
; is no response by then, you will declare the response lost.

RECV:
        $QIOW_S FUNC=#IO$_READVBLK!IO$M_NOW,CHAN=DEVCHAN,IOSB=IOSB,-
                P1=RCVBUF,P2=#RCVBUFLEN,P5=#RCVP5
        BLBS    R0,RECV_REQ_OK
        BRW     ERROR

RECV_REQ_OK:

        MOVZWL  IOSB,R0
        BLBS    R0,RECV_IO_OK
        CMPW    R0,#SS$_ENDOFFILE       ; Was there just no message available?
        BEQL    10$                     ; Branch if so to try again
        BRW     ERROR

; If you are able to post 1000 reads and not receive the response packet, then
; you will assume the packet is lost.

10$:    CMPW    TRY,#1000               ; Have you tried enough?
        BGTR    LOST                    ; If GTR, yes, so message is lost
        INCW    TRY                     ; Try again
        BRB     RECV


RECV_IO_OK:

; You received a message.  Check that the Source Address matches the place we
; sent the message.

        CMPL    XMTP5,RCVSA
        BNEQ    RECV_BAD
        CMPW    XMTP5+4,RCVSA+4
        BEQL    RECV_OK

; There was something wrong with the message received.

RECV_BAD:
        PUSHAB  BMSG
        BRB     EXIT

; The test went fine.  Print a success message.
```

(continued on next page)

Example 8–5 (Cont.) LANETH.MAR Local Area Network Programming Example

```
RECV_OK:
        PUSHAB  GMSG
        BRB     EXIT

; You lost the message.  Print a message stating so.

LOST:
        PUSHAB  LMSG
        BRB     EXIT

; There was an error while running the test.  Print a message stating so.

ERROR:
        PUSHAB  EMSG
        BRB     EXIT

; The test is done.  You will call LIB$PUT_OUTPUT to display the status of
; this test.  The message that will be displayed has its descriptor on the
; stack.  That descriptor will be used by the LIB$PUT_OUTPUT routine.

EXIT:
        CALLS   #1,G^LIB$PUT_OUTPUT
        $EXIT_S

        .END    START
```

Example 8–6 LAN802.MAR Local Area Network Programming Example

```
        .TITLE  EXAMPLE                 802 SAMPLE TEST PROGRAM
        .IDENT  /X01/

; This 802 test program will send a TEST message to another system and
; wait for a response.  Since you will be sending the message to the
; MAC Sublayer on the other node, you should always get a response as
; long as the other node exists.
;
; Note that this test will try to use the device defined by the logical
; LAN as the LAN device.  If this does not work, then it will try to use
; one of the currently known LAN devices.  To use a particular Ethernet
; or FDDI device, you need to define the logical LAN to be the name of the
; device you wish t use.  For example:
;
;       $ DEFINE LAN EXB0

        .LIBRARY        "SYS$LIBRARY:LIB.MLB"

        $IODEF                          ; Define I/O functions and modifiers
        $NMADEF                         ; Define Network Management parameters

; Local definitions

        RCVBUFLEN = 512                 ; Size of receive buffer
        XMTBUFLEN = 20                  ; Size of transmit buffer

; Setmode parameter buffer.  For 802, you are required to state the packet
; format and our unique SAP value.


SETPARM:
        .WORD   NMA$C_PCLI_FMT          ; Packet format
                .LONG   NMA$C_LINFM_802
        .WORD   NMA$C_PCLI_SAP          ; Our individual SAP address
                .LONG   2

        SETPARMLEN = .-SETPARM

SETPARMDSC:
        .LONG           SETPARMLEN
        .ADDRESS        SETPARM

; P2 transmit data buffer

XMTBUF:
        .BYTE   00,01,02,03,04,05,06,07,08,09
        .BYTE   10,11,12,13,14,15,16,17,18,19

; P4 transmit DSAP and CTL field values

XMTP4:
        .BYTE   0                       ; DSAP for transmit is the MAC
                                        ; Sublayer SAP (zero)
        .WORD   NMA$C_CTLVL_TEST        ; The CTL field value is TEST

; P4 transmit descriptor
```

(continued on next page)

Example 8–6 (Cont.) LAN802.MAR Local Area Network Programming Example

```
XMTP4DSC:
        .LONG           3               ; P4 is always 3 bytes in size
        .ADDRESS        XMTP4           ; Address of buffer

; P5 transmit destination address
;
; Set this value to be a node on your LAN that supports 802 packet
; format.

XMTP5:
        .BYTE   ^XAA,^X00,^X04,^X00,^X17,^X4E

; P2 receive data buffer

RCVBUF:
        .BLKB   RCVBUFLEN

; P5 receive header buffer

RCVP5:
RCVDA:  .BLKB   6
RCVSA:  .BLKB   6
RCVDSAP:.BLKB   1
RCVSSAP:.BLKB   1
RCVCTL: .BLKB   2

; Messages used to display status of this program.

GMSG:   .ASCID  "Successful test"
BMSG:   .ASCID  "Received packet was not what was expected"
LMSG:   .ASCID  "Packet lost or node not responding"
EMSG:   .ASCID  "Error occurred while running test"
DMSG:   .ASCID  "No LAN device found - please define LAN correctly"


; Miscellaneous data structures

TRY:    .WORD   0                       ; Number of times you have tried
                                        ; the READ QIO (start at 0)
IOSB:   .BLKQ   1                       ; I/O status block
LANDSC1:.ASCID  'LAN'                   ; Units to use for test
LANDSC2:.ASCID  'ECA0'
LANDSC3:.ASCID  'ESA0'
LANDSC4:.ASCID  'ETA0'
LANDSC5:.ASCID  'EXA0'
LANDSC6:.ASCID  'EZA0'
LANDSC7:.ASCID  'FCA0'
LANDSC8:.ASCID  'FXA0'
LANDSC9:.ASCID  'XEA0'
LANDSCA:.ASCID  'XQA0'
DSCADR: .ADDRESS        LANDSC1         ; Table of addresses pointing to
        .ADDRESS        LANDSC2         ; the descriptors of device names
        .ADDRESS        LANDSC3
        .ADDRESS        LANDSC4
        .ADDRESS        LANDSC5
        .ADDRESS        LANDSC6
        .ADDRESS        LANDSC7
        .ADDRESS        LANDSC8
        .ADDRESS        LANDSC9
        .ADDRESS        LANDSCA
        .LONG           0
DEVCHAN:.BLKL   1                       ; Returned port number
```

(continued on next page)

Example 8–6 (Cont.) LAN802.MAR Local Area Network Programming Example

```
;*****************************************************************************
;
; Start of code
;
;*****************************************************************************

        .ENTRY  START,^M<>

; Assign a port to the LAN device.  If LAN does not work, try each
; of the currently known LAN devices.

ASSIGN:
        MOVAL   DSCADR,R5               ; Start at beginning
10$:    MOVL    (R5),R4                 ; Is there a descriptor?
        BEQL    20$                     ; If EQL, no
        $ASSIGN_S DEVNAM=(R4),CHAN=DEVCHAN
        BLBS    R0,ASSIGN_OK
        ADDL    #4,R5                   ; Skip to next descriptor
        CMPW    R0,#SS$_NOSUCHDEV       ; Was the device not there?
        BEQL    10$                     ; If EQL, yes, try next device
        BRW     ERROR

; You could not find a LAN device to assign a port to.

20$:    PUSHAB  DMSG
        BRW     EXIT

ASSIGN_OK:

; Set up the port's characteristics.

        $QIOW_S FUNC=#<IO$_SETMODE!IO$M_CTRL!IO$M_STARTUP>,-
                CHAN=DEVCHAN,IOSB=IOSB,-
                P2=#SETPARMDSC
        BLBS    R0,STARTUP_REQ_OK
        BRW     ERROR

STARTUP_REQ_OK:

        MOVZWL  IOSB,R0
        BLBS    R0,STARTUP_IO_OK
        BRW     ERROR

STARTUP_IO_OK:

; Now transmit our TEST message.

        $QIOW_S FUNC=#IO$_WRITEVBLK,CHAN=DEVCHAN,IOSB=IOSB,-
                P1=XMTBUF,P2=#XMTBUFLEN,P4=#XMTP4DSC,P5=#XMTP5
        BLBS    R0,XMIT_REQ_OK
        BRW     ERROR


XMIT_REQ_OK:

        MOVZWL  IOSB,R0
        BLBS    R0,XMIT_IO_OK
        BRW     ERROR

XMIT_IO_OK:

; Now try to receive the response.  You will use the IO$M_NOW function modifier
; on the READ so that you don't hang here waiting forever if there is no
; response.  You will attempt to receive the message 1000 times.  If there
; is no response by then, you will declare the response lost.
```

(continued on next page)

Example 8–6 (Cont.) LAN802.MAR Local Area Network Programming Example

```
RECV:
        $QIOW_S FUNC=#IO$_READVBLK!IO$M_NOW,CHAN=DEVCHAN,IOSB=IOSB,-
                P1=RCVBUF,P2=#RCVBUFLEN,P5=#RCVP5
        BLBS    R0,RECV_REQ_OK
        BRW     ERROR

RECV_REQ_OK:

        MOVZWL  IOSB,R0
        BLBS    R0,RECV_IO_OK
        CMPW    R0,#SS$_ENDOFFILE       ; Was there just no message available?
        BEQL    10$                     ; Branch if so to try again
        BRW     ERROR

; If you are able to post 1000 reads and not receive the response packet, then
; you will assume the packet is lost.

10$:    CMPW    TRY,#1000               ; Have you tried enough?
        BGTR    LOST                    ; If GTR, yes, so message is lost
        INCW    TRY                     ; Try again
        BRB     RECV

RECV_IO_OK:

; You received a message.  Check that the Source Address matches the place we
; sent the message.

        CMPL    XMTP5,RCVSA
        BNEQ    RECV_BAD
        CMPW    XMTP5+4,RCVSA+4
        BNEQ    RECV_BAD


; Check that the data received was the correct size.

        CMPW    #XMTBUFLEN,IOSB+2
        BNEQ    RECV_BAD

; Check that the data received matches the data you sent.

        MOVZBL  #XMTBUFLEN,R0
        MOVAB   XMTBUF,R1
        MOVAB   RCVBUF,R2
10$:    CMPB    (R1)+,(R2)+
        BNEQ    RECV_BAD
        SOBGTR  R0,10$
        BRB     RECV_OK                 ; All bytes matched

; There was something wrong with the message received.

RECV_BAD:
        PUSHAB  BMSG
        BRB     EXIT

; The test went fine.  Print a success message.

RECV_OK:
        PUSHAB  GMSG
        BRB     EXIT

; You lost the message.  Print a message stating so.

LOST:
        PUSHAB  LMSG
        BRB     EXIT

; There was an error while running the test.  Print a message stating so.
```

(continued on next page)

Example 8–6 (Cont.) LAN802.MAR Local Area Network Programming Example

```
ERROR:
        PUSHAB  EMSG
        BRB     EXIT

; The test is done.  You will call LIB$PUT_OUTPUT to display the status of
; this test.  The message that will be displayed has its descriptor on the
; stack.  That descriptor will be used by the LIB$PUT_OUTPUT routine.

EXIT:
        CALLS   #1,G^LIB$PUT_OUTPUT
        $EXIT_S

        .END    START
```

8.11 References

The following publications provide more information on local area networks.

- IEEE Standards 802.1 (A, B, C, and D), 802.2, and 802.3.
- *The Ethernet–Data Link Layer and Physical Layer Specification*
- ANSI X3t9.5 and X3.139
- *Digital FDDI Network Architecture*

A
I/O Function Codes

This appendix lists the function codes and function modifiers defined in the $IODEF macro. The arguments for these functions are also listed.

A.1 ACP-QIO Interface Driver

This section lists the function codes and function modifiers for the ACP-QIO interface driver.

Functions	Arguments	Modifiers
IO$_CREATE IO$_ACCESS IO$_DEACCESS IO$_MODIFY IO$_DELETE IO$_ACPCONTROL	P1—FIB descriptor address P2—file name string address P3—result string length address P4—result string descriptor address P5—attribute list address	IO$M_CREATE[1] IO$M_ACCESS[1] IO$M_DELETE[2] IO$M_DMOUNT[3]
IO$_MOUNT	None	None

[1]Only for IO$_CREATE and IO$_ACCESS

[2]Only for IO$_CREATE and IO$_DELETE

[3]Only for IO$_ACPCONTROL

QIO Status Returns		
SS$_ACCONFLICT	SS$_ACPVAFUL	SS$_BADATTRIB
SS$_BADCHKSUM	SS$_BADFILEHDR	SS$_BADFILENAME
SS$_BADFILEVER	SS$_BADIRECTORY	SS$_BADPARAM
SS$_BADQFILE	SS$_BLOCKCNTERR	SS$_CREATED
SS$_DEVICEFULL	SS$_DIRFULL	SS$_DIRNOTEMPTY
SS$_DUPDSKQUOTA	SS$_DUPFILENAME	SS$_ENDOFFILE
SS$_EXBYTLM	SS$_EXDISKQUOTA	SS$_FCPREADERR
SS$_FCPREWNDERR	SS$_FCPSPACERR	SS$_FCPWRITERR
SS$_FILELOCKED	SS$_FILENUMCHK	SS$_FILEPURGED
SS$_FILESEQCHK	SS$_FILESTRUCT	SS$_FILNOTEXP
SS$_HEADERFULL	SS$_IBCERROR[1]	SS$_IDXFILEFULL

[1]The second longword of the IOSB contains a job controller status code.

QIO Status Returns		
SS$_ILLCNTRFUNC	SS$_NODISKQUOTA	SS$_NOMOREFILES
SS$_NOPRIV	SS$_NOQFILE	SS$_NOSUCHFILE
SS$_NOTAPEOP	SS$_NOTLABELMT	SS$_NOTPRINTED[1]
SS$_NOTVOLSET	SS$_OVRDSKQUOTA	SS$_QFACTIVE
SS$_QFNOTACT	SS$_SERIOUSEXCP	SS$_SUPERSEDE
SS$_TAPEPOSLOST	SS$_TOOMANYVER	SS$_WRITLCK
SS$_WRONGACP		

[1]The second longword of the IOSB contains a job controller status code.

A.2 Disk Drivers

This section lists the function codes and function modifiers for the disk drivers.

Functions	Arguments	Modifiers
IO$_READVBLK IO$_READLBLK IO$_READPBLK[4] IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK[4]	P1—buffer address P2—byte count P3—disk address	IO$M_INHSEEK[1] IO$M_DATACHECK[2] IO$M_DELDATA[3] IO$M_INHRETRY IO$M_ERASE[5]
IO$_WRITECHECK[2]	P1—buffer address P2—byte count P3—disk address	None
IO$_SENSECHAR IO$_SENSEMODE IO$_PACKACK IO$_AVAILABLE IO$_UNLOAD	None	None
IO$_SEARCH	P1—read/write head position	None
IO$_SEEK[4]	P1—seek to specified cylinder	None
IO$_FORMAT	P1—RX02 density	None

[1]Only for IO$READPBLK and IO$_WRITEPBLK (not for TU58, RX01, RX02, RB02, or RL02)

[2]Not for RX01 and RX02

[3]Only for IO$_WRITEPBLK on RX02

[4]Not for DSA disks

[5]Only for write functions

Functions	Arguments	Modifiers
IO$_CREATE IO$_ACCESS IO$_DEACCESS IO$_MODIFY IO$_DELETE IO$_ACPCONTROL	P1—FIB descriptor address P2—file name string address P3—result string length address P4—result string descriptor address P5—attribute list address	IO$M_CREATE[6] IO$M_ACCESS[6] IO$M_DELETE[7] IO$M_DMOUNT[8]

[6]Only for IO$_CREATE and IO$_ACCESS

[7]Only for IO$_CREATE and IO$_DELETE

[8]Only for IO$_ACPCONTROL

QIO Status Returns		
SS$_ABORT	SS$_CANCEL	SS$_CTRLERR
SS$_DATACHECK	SS$_DATAOVERUN	SS$_DRVERR
SS$_FORCEDERR	SS$_FORMAT	SS$_ILLIOFUNC
SS$_IVADDR	SS$_IVBUFLEN	SS$_MEDOFL
SS$_NONEXDRV	SS$_NORMAL	SS$_OPINCOMPL
SS$_PARITY	SS$_RCT	SS$_RDDELDATA
SS$_TIMEOUT	SS$_UNSAFE	SS$_VOLINV
SS$_WASECC	SS$_WRITLCK	

A.3 Magnetic Tape Drivers

This section lists the function codes and function modifiers for the magnetic tape drivers.

Functions	Arguments	Modifiers
IO$_READVBLK IO$_READLBLK IO$_READPBLK	P1—buffer address P2—byte count	IO$M_DATACHECK[1] IO$M_INHRETRY IO$M_REVERSE[3]
IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1—buffer address P2—byte count	IO$M_DATACHECK[1] IO$M_INHRETRY IO$M_INHEXTGAP[2] IO$M_NOWAIT[8] IO$M_ERASE[7]
IO$_SETMODE IO$_SETCHAR	P1—characteristics buffer address P2—characteristics buffer length[9]	

[1]Not for TS04 and TU80

[2]Only for TE16, TU45, and TU77

[3]Not for TK50

[7]IO$M_ERASE takes no arguments; only for IO$_WRITELBLK and IO$_WRITEPBLK on TMSCP drives.

[8]Only for TU81-Plus drives

[9]Only for TMSCP drives

Functions	Arguments	Modifiers
IO$_CREATE IO$_ACCESS IO$_DEACCESS IO$_MODIFY IO$_ACPCONTROL	P1—FIB descriptor address P2—file name string address P3—result string length address P4—result string descriptor address P5—attribute list address	IO$M_CREATE[4] IO$M_ACCESS[4] IO$M_DMOUNT[5]
IO$_SKIPFILE	P1—skip n tape marks	IO$M_INHRETRY IO$M_NOWAIT[8]
IO$_SKIPRECORD	P1—skip n blocks	IO$M_INHRETRY IO$M_NOWAIT[8]
IO$_REWIND IO$_REWINDOFF IO$_UNLOAD	None	IO$M_INHRETRY IO$M_NOWAIT IO$M_RETENSION
IO$_WRITEOF	None	IO$M_INHEXTGAP[2] IO$M_INHRETRY IO$M_NOWAIT[8]
IO$_SENSEMODE IO$_SENSECHAR	P1—characteristics buffer address[9] P2—characteristics buffer length[9]	IO$M_INHRETRY
IO$_DSE[6] IO$_PACKACK IO$_AVAILABLE	None	None

[2]Only for TE16, TU45, and TU77

[4]Only for IO$_CREATE and IO$_ACCESS

[5]Only for IO$_ACPCONTROL

[6]Only for TU78, TU81, TA81, and TA78

[8]Only for TU81-Plus drives

[9]Only for TMSCP drives

QIO Status Returns		
SS$_ABORT	SS$_CANCEL	SS$_CTRLERR
SS$_DATACHECK	SS$_DATAOVERUN	SS$_DEVOFFLINE
SS$_DRVERR	SS$_ENDOFFILE	SS$_ENDOFTAPE
SS$_ENDOFVOLUME	SS$_FORMAT	SS$_ILLIOFUNC
SS$_MEDOFL	SS$_NONEXDRV	SS$_NORMAL
SS$_OPINCOMPL	SS$_PARITY	SS$_SERIOUSEXCP
SS$_TIMEOUT	SS$_UNSAFE	SS$_VOLINV
SS$_WRITLCK		

A.4 Mailbox Driver

This section lists the function codes and function modifiers for the mailbox driver.

Functions	Arguments	Modifiers
IO$_READVBLK IO$_READLBLK IO$_READPBLK IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1—buffer address P2—buffer size	IO$M_NOW IO$M_NORSWAIT[1] IO$M_ READERCHECK[1] IO$M_ WRITERCHECK[2] IO$M_STREAM[2]
IO$_WRITEOF	None	IO$M_NOW IO$M_ READERCHECK IO$M_STREAM
IO$_SETMODE!IO$M_READATTN IO$_SETMODE!IO$M_WRTATTN IO$_SETMODE!IO$MB_ROOM_ NOTIFY	P1—AST address P2—AST parameter P3—access mode	None
†IO$_SETMODE!IO$M_ READERWAIT †IO$_SETMODE!IO$M_ WRITERWAIT	None	None
IO$_SETMODE!IO$M_SETPROT	P2—volume protection mask	None
†IO$_SENSEMODE!IO$M_ READERCHECK †IO$_SENSEMODE!IO$M_ WRITERCHECK	None	None

[1]Only for write functions

[2]Only for read functions

†VAX specific

QIO Status Returns in R0			
SS$_ACCVIO	SS$_EXQUOTA	SS$_ILLIOFUNC	SS$INSFMEM
SS$MBFULL	SS$_MBTOOSML	SS$_NOPRIV	SS$_NORMAL

IOSB Status Returns			
SS$_ABORT	SS$_BUFFEROVF	SS$_CANCEL	SS$_ENDOFFILE
SS$_ NOREADER	SS$_NORMAL	SS$_NOWRITER	

A.5 Terminal Driver

This section lists the function codes and function modifiers for the terminal driver.

Functions	Arguments	Modifiers
IO$_READVBLK IO$_READLBLK IO$_READPROMPT	P1—buffer address P2—buffer size P3—timeout P4—read terminator block address P5—prompt string buffer address P6—prompt string buffer size[1]	IO$M_NOECHO IO$M_CVTLOW IO$M_NOFILTR IO$M_TIMED IO$M_PURGE IO$M_DSABLMBX IO$M_TRMNOECHO IO$M_ESCAPE
IO$_READVBLK	P1—buffer address P2—buffer size P3—access mode to probe itemlist P4—(zero) P5—itemlist buffer address P6—itemlist buffer size	IO$M_EXTEND[2]
IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1—buffer address P2—buffer size P3—(ignored) P4—carriage control specifier[3]	IO$M_CANCTRLO IO$M_ENABLMBX IO$M_NOFORMAT IO$M_REFRESH IO$M_BREAKTHRU
IO$_SETMODE IO$_SETCHAR	P1—characteristics buffer address P2—characteristics buffer size P3—speed specifier P4—fill specifier P5—parity flags	
IO$_SETMODE IO$_SETCHAR	None	IO$M_HANGUP
IO$_SETMODE	P1—buffer address P2—buffer size	IO$M_BRDCST
IO$_SETMODE IO$_SETCHAR	P1—AST service routine address P2—AST parameter P3—access mode to deliver AST	IO$M_CTRLCAST IO$M_CTRLYAST
IO$_SETMODE IO$_SETCHAR	P1—AST service routine address P2—character mask address P3—access mode to deliver AST	IO$M_OUTBAND IO$M_TT_ABORT[4] IO$M_INCLUDE[4]

[1]Only for IO$_READPROMPT

[2]Only for itemlist read function. Do not specify with other modifiers.

[3]Only for IO$_WRITELBLK and IO$_WRITEVBLK

[4]Only with IO$M_OUTBAND

Functions	Arguments	Modifiers
IO$_SETMODE IO$_SETCHAR	P1—address of control signals	IO$M_SET_MODEM[5] IO$M_MAINT
IO$_SETMODE IO$_SETCHAR	None	IO$M_LOOP[5] IO$M_UNLOOP[5] IO$M_MAINT
IO$_TTY_PORT		IO$M_LT_CONNECT IO$M_LT_DISCON
IO$_TTY_PORT	P1—itemlist[6] address P2—queued status	IO$M_LT_MAP_PORT
IO$_TTY_PORT	P1—service name descriptor address P2—service rating	IO$M_LT_RATING
IO$_TTY_PORT	P1—itemlist address P2—itemlist length P3—entity type P4—entity string descriptor	IO$M_LT_SENSEMODE
IO$_TTY_PORT	P1—itemlist address P2—itemlist length P3—entity type P4—entity string descriptor	IO$M_LT_SETMODE
IO$_SENSEMODE IO$_SENSECHAR	P1—characteristics buffer address P2—characteristics buffer size	IO$M_TYPEAHDCNT
IO$_SENSEMODE IO$_SENSECHAR	P1—address of input modem signal block	IO$M_RD_MODEM
IO$_SENSEMODE	P1—buffer address P2—buffer size	IO$M_BRDCST

[5]Only with IO$M_MAINT

[6]Itemlist: IO$V_LT_MAP_NODNAM, IO$V_LT_MAP_PORNAM, IO$V_LT_MAP_SRVNAM, IO$V_LT_MAP_LNKNAM, and IO$V_LT_MAP_NETADR.

QIO Status Returns		
SS$_ABORT	SS$_BADESCAPE	SS$_BADPARAM
SS$_CANCEL	SS$_CHANINTLK	SS$_CONTROLC
SS$_CONTROLO	SS$_CONTROLY	SS$_DATAOVERUN
SS$_INCOMPAT	SS$_NORMAL	SS$_PARITY
SS$_PARTESCAPE	SS$_TIMEOUT	

A.6 Local Area Network Device Drivers

This section lists the function codes and function modifiers for the local area network drivers.

Functions	Arguments	Modifiers
IO$_READLBLK IO$_READVBLK IO$_READPBLK IO$_WRITELBLK IO$_WRITEVBLK IO$_WRITEPBLK	P1—buffer address P2—buffer size P4—802 format fields (optional)[3] P5—destination address (optional)[3]	IO$M_NOW[1] IO$M_RESPONSE[2]
IO$_SETMODE IO$_SETCHAR	P2—extended characteristics buffer (optional)[4]	IO$M_CTRL IO$M_STARTUP IO$M_SHUTDOWN
IO$_SETMODE IO$_SETCHAR	P1—AST service address P3 – access mode to deliver AST	IO$M_ATTNAST
IO$_SENSEMODE IO$_SENSECHAR	P1—device characteristics buffer (optional) P2—extended characteristics buffer (optional)	IO$M_CTRL

[1]Only for read functions

[2]Only for write functions

[3]See text for complete contents

[4]Use only with IO$M_CTRL alone or with IO$_STARTUP, that is, the set controller mode

QIO Status Returns		
SS$_ABORT	SS$_ACCVIO	SS$_BADPARAM
SS$_BUFFEROVF	SS$_COMMHARD	SS$_CTRLERR
SS$_DATACHECK	SS$_DATAOVERUN	SS$_DEVACTIVE
SS$_DEVALLOC	SS$_DEVINACT	SS$_DEVOFFLINE
SS$_DEVREQERR	SS$_DISCONNECT	SS$_DUPUNIT
SS$_ENDOFFILE	SS$_EXQUOTA	SS$_INSFMEM
SS$_INSFMAPREG	SS$_IVBUFLEN	SS$_MEDOFL
SS$_NOPRIV	SS$_NORMAL	SS$_OPINCOMPL
SS$_TIMEOUT	SS$_TOOMUCHDATA	

B
Tables

This appendix includes tables for the DEC Multinational character set and for terminal escape sequences and modes.

B.1 DEC Multinational Character Set

Table B–1 lists the DEC Multinational character set. The DEC Multinational character set is an 8-bit character set with 256 characters. The first 128 characters in the set correspond to the ASCII character set. The *VAX EDT Reference Manual* lists the graphics for these characters and describes how to enter them from various types of terminals.

Table B–1 DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Control Characters[1]		
00	000	000	NUL	null character
01	001	001	SOH	start of heading (Ctrl/A)
02	002	002	STX	start of text (Ctrl/B)
03	003	003	ETX	end of text (Ctrl/C)
04	004	004	EOT	end of transmission (Ctrl/D)
05	005	005	ENQ	enquiry (Ctrl/E)
06	006	006	ACK	acknowledge (Ctrl/F)
07	007	007	BEL	bell (Ctrl/G)
08	010	008	BS	backspace (Ctrl/H)
09	011	009	HT	horizontal tabulation (Ctrl/I)
0A	012	010	LF	line feed (Ctrl/J)
0B	013	011	VT	vertical tabulation (Ctrl/K)
0C	014	012	FF	form feed (Ctrl/L)
0D	015	013	CR	carriage return (Ctrl/M)
0E	016	014	SO	shift out (Ctrl/N)
0F	017	015	SI	shift in (Ctrl/O)
10	020	016	DLE	data link escape (Ctrl/P)
11	021	017	DC1	device control 1 (Ctrl/Q)
12	022	018	DC2	device control 2 (Ctrl/R)

[1]The ALTMODE and DELETE characters (decimal 125, 126, and 127) are also control characters.

(continued on next page)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Control Characters[1]		
13	023	019	DC3	device control 3 (Ctrl/S)
14	024	020	DC4	device control 4 (Ctrl/T)
15	025	021	NAK	negative acknowlege (Ctrl/U)
16	026	022	SYN	synchronous idle (Ctrl/V)
17	027	023	ETB	end of transmission block (Ctrl /W)
18	030	024	CAN	cancel (Ctrl/X)
19	031	025	EM	end of medium (Ctrl/Y)
1A	032	026	SUB	substitute (Ctrl/Z)
1B	033	027	ESC	escape
1C	034	028	FS	file separator
1D	035	029	GS	group separator
1E	036	030	RS	record separator
1F	037	031	US	unit separator
		ASCII Special and Numeric Characters		
20	040	032	SP	space
21	041	033	!	exclamation point
22	042	034	"	quotation marks (double quote)
23	043	035	#	number sign
24	044	036	$	dollar sign
25	045	037	%	percent sign
26	046	038	&	ampersand
27	047	039	′	apostrophe (single quote)
28	050	040	(	opening parenthesis
29	051	041	)	closing parenthesis
2A	052	042	*	asterisk
2B	053	043	+	plus
2C	054	044	,	comma
2D	055	045	–	hyphen or minus
2E	056	046	.	period or decimal point
2F	057	047	/	slash
30	060	048	0	zero
31	061	049	1	one
32	062	050	2	two
33	063	051	3	three

[1]The ALTMODE and DELETE characters (decimal 125, 126, and 127) are also control characters.

(continued on next page)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
ASCII Special and Numeric Characters				
34	064	052	4	four
35	065	053	5	five
36	066	054	6	six
37	067	055	7	seven
38	070	056	8	eight
39	071	057	9	nine
3A	072	058	:	colon
3B	073	059	;	semicolon
3C	074	060	<	less than
3D	075	061	=	equals
3E	076	062	>	greater than
3F	077	063	?	question mark
ASCII Alphabetic Characters				
40	100	064	@	commercial at sign
41	101	065	A	uppercase A
42	102	066	B	uppercase B
43	103	067	C	uppercase C
44	104	068	D	uppercase D
45	105	069	E	uppercase E
46	106	070	F	uppercase F
47	107	071	G	uppercase G
48	110	072	H	uppercase H
49	111	073	I	uppercase I
4A	112	074	J	uppercase J
4B	113	075	K	uppercase K
4C	114	076	L	uppercase L
4D	115	077	M	uppercase M
4E	116	078	N	uppercase N
4F	117	079	O	uppercase O
50	120	080	P	uppercase P
51	121	081	Q	uppercase Q
52	122	082	R	uppercase R
53	123	083	S	uppercase S
54	124	084	T	uppercase T
55	125	085	U	uppercase U

(continued on next page)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alphabetic Characters		
56	126	086	V	uppercase V
57	127	087	W	uppercase W
58	130	088	X	uppercase X
59	131	089	Y	uppercase Y
5A	132	090	Z	uppercase Z
5B	133	091	[	left bracket
5C	134	092	\	backslash
5D	135	093	]	right bracket
5E	136	094	^	circumflex
5F	137	095	_	underscore
60	140	096	‘	grave accent
61	141	097	a	lowercase a
62	142	098	b	lowercase b
63	143	099	c	lowercase c
64	144	100	d	lowercase d
65	145	101	e	lowercase e
66	146	102	f	lowercase f
67	147	103	g	lowercase g
68	150	104	h	lowercase h
69	151	105	i	lowercase i
6A	152	106	j	lowercase j
6B	153	107	k	lowercase k
6C	154	108	l	lowercase l
6D	155	109	m	lowercase m
6E	156	110	n	lowercase n
6F	157	111	o	lowercase o
70	160	112	p	lowercase p
71	161	113	q	lowercase q
72	162	114	r	lowercase r
73	163	115	s	lowercase s
74	164	116	t	lowercase t
75	165	117	u	lowercase u
76	166	118	v	lowercase v
77	167	119	w	lowercase w
78	170	120	x	lowercase x
79	171	121	y	lowercase y

(continued on next page)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alphabetic Characters		
7A	172	122	z	lowercase z
7B	173	123	{	left brace
7C	174	124	|	vertical line
7D	175	125	}	right brace (ALTMODE)
7E	176	126	~	tilde (ALTMODE)
7F	177	127	DEL	rubout (DELETE)
80	200	128	—	[reserved]
81	201	129	—	[reserved]
82	202	130	—	[reserved]
83	203	131	—	[reserved]
84	204	132	IND	index
85	205	133	NEL	next line
86	206	134	SSA	start of selected area
87	207	135	ESA	end of started area
88	210	136	HTS	horizontal tab set
89	211	137	HTJ	horizontal tab set with justification
8A	212	138	VTS	vertical tab set
8B	213	139	PLD	partial line down
8C	214	140	PLU	partial line up
8D	215	141	RI	reverse index
8E	216	142	SS2	single shift 2
8F	217	143	SS3	single shift 3
90	220	144	DCS	device control string
91	221	145	PU1	private use 1
92	222	146	PU2	private use 2
93	223	147	STS	set transmit state
94	224	148	CCH	cancel character
95	225	149	MW	message waiting
96	226	150	SPA	start of protected area
97	227	151	EPA	end of protected area
98	230	152	—	[reserved]
99	231	153	—	[reserved]
9A	232	154	—	[reserved]
9B	233	155	CSI	control sequence introducer
9C	234	156	ST	string terminator
9D	235	157	OSC	operating system command

(continued on next page)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alphabetic Characters		
9E	236	158	PM	privacy message
9F	237	159	APC	application
A0	240	160	—	[reserved]
A1	241	161	¡	inverted exclamation point
A2	242	162	¢	cent sign
A3	243	163	£	pound sign
A4	244	164	—	[reserved]
A5	245	165	¥	yen sign
A6	246	166	—	[reserved]
A7	247	167	§	section sign
A8	250	168	¤	general currency sign
A9	251	169	©	copyright sign
AA	252	170	ª	feminine ordinal indicator
AB	253	171	«	angle quotation mark left
AC	254	172	—	[reserved]
AD	255	173	—	[reserved]
AE	256	174	—	[reserved]
AF	257	175	—	[reserved]
B0	260	176	°	degree sign
B1	261	177	±	plus/minus sign
B2	262	178	²	superscript 2
B3	263	179	³	superscript 3
B4	264	180	—	[reserved]
B5	265	181	µ	micro sign
B6	266	182	¶	paragraph sign, pilcrow
B7	267	183	·	middle dot
B8	270	184	—	[reserved]
B9	271	185	¹	superscript 1
BA	272	186	º	masculine ordinal indicator
BB	273	187	»	angle quotation mark right
BC	274	188	¼	fraction one-quarter
BD	275	189	½	fraction one-half
BE	276	190	—	[reserved]
BF	277	191	¿	inverted question mark
C0	300	192	À	uppercase A with grave accent

(continued on next page)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alphabetic Characters		
C1	301	193	Á	uppercase A with acute accent
C2	302	194	Â	uppercase A with circumflex
C3	303	195	Ã	uppercase A with tilde
C4	304	196	Ä	uppercase A with umlaut, (diaeresis)
C5	305	197	Å	uppercase A with ring
C6	306	198	Æ	uppercase AE diphthong
C7	307	199	Ç	uppercase C with cedilla
C8	310	200	È	uppercase E with grave accent
C9	311	201	É	uppercase E with acute accent
CA	312	202	Ê	uppercase E with circumflex
CB	313	203	Ë	uppercase E with umlaut, (diaeresis)
CC	314	204	Ì	uppercase I with grave accent
CD	315	205	Í	uppercase I with acute accent
CE	316	206	Î	uppercase I with circumflex
CF	317	207	Ï	uppercase I with umlaut, (diaeresis)
D0	320	208	—	[reserved]
D1	321	209	Ñ	uppercase N with tilde
D2	322	210	Ò	uppercase O with grave accent
D3	323	211	Ó	uppercase O with acute accent
D4	324	212	Ô	uppercase O with circumflex
D5	325	213	Õ	uppercase O with tilde
D6	326	214	Ö	uppercase O with umlaut, (diaeresis)
D7	327	215	Œ	uppercase OE ligature
D8	330	216	Ø	uppercase O with slash
D9	331	217	Ù	uppercase U with grave accent
DA	332	218	Ú	uppercase U with acute accent
DB	333	219	Û	uppercase U with circumflex

(continued on next page)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alphabetic Characters		
DC	334	220	Ü	uppercase U with umlaut, (diaeresis)
DD	335	221	Ÿ	uppercase Y with umlaut, (diaeresis)
DE	336	222	—	[reserved]
DF	337	223	ß	German lowercase sharp s
E0	340	224	à	lowercase a with grave accent
E1	341	225	á	lowercase a with acute accent
E2	342	226	â	lowercase a with circumflex
E3	343	227	ã	lowercase a with tilde
E4	344	228	ä	lowercase a with umlaut, (diaeresis)
E5	345	229	å	lowercase a with ring
E6	346	230	æ	lowercase ae diphthong
E7	347	231	ç	lowercase c with cedilla
E8	350	232	è	lowercase e with grave accent
E9	351	233	é	lowercase e with acute accent
EA	352	234	ê	lowercase e with circumflex
EB	353	235	ë	lowercase e with umlaut, (diaeresis)
EC	354	236	ì	lowercase i with grave accent
ED	355	237	í	lowercase i with acute accent
EE	356	238	î	lowercase i with circumflex
EF	357	239	ï	lowercase i with umlaut, (diaeresis)
F0	360	240	—	[reserved]
F1	361	241	ñ	lowercase n with tilde
F2	362	242	ò	lowercase o with grave accent
F3	363	243	ó	lowercase o with acute accent
F4	364	244	ô	lowercase o with circumflex
F5	365	245	õ	lowercase o with tilde
F6	366	246	ö	lowercase o with umlaut, (diaeresis)

(continued on next page)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alphabetic Characters		
F7	367	247	œ	lowercase oe ligature
F8	370	248	ø	lowercase o with slash
F9	371	249	ù	lowercase u with grave accent
FA	372	250	ú	lowercase u with acute accent
FB	373	251	û	lowercase u with circumflex
FC	374	252	ü	lowercase u with umlaut, (diaeresis)
FD	375	253	ÿ	lowercase y with umlaut, (diaeresis)
FE	376	254	—	[reserved]
FF	377	255	—	[reserved]

B.2 Terminal Sequences and Modes

Table B–2 lists the valid ANSI and Digital-private escape sequences for terminals that have the TT2$M_ANSICRT, TT2$M_DECCRT, TT2$M_AVO, TT2$M_EDIT, and TT2$M_BLOCK characteristics (see Section 5.2.1.4). Table B–2 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 VT100-, VT200-, or VT300- family user's guide). Unless otherwise noted, the operation of escape sequences and modes is identical to the particular VT100-, VT200-, or VT300-family terminals that implement these features.

Table B–2 Sequences and Modes

Name	Valid Parameters	ANSICRT	DECCRT	AVO	EDIT	BLOCK[1]
		ANSI-Defined Escape Sequences				
CPR	All	x	x			
CUB	All	x	x			
CUD	All	x	x			
CUF	All	x	x			
CUP	All	x	x			
CUU	All	x	x			
DSR	0,3,5,6	x	x			
ED	0,1,2	x	x			
EL	0,1,2	x	x			

[1]Terminal characteristics. Prefix is TT2$M_.

(continued on next page)

Table B–2 (Cont.) Sequences and Modes

Name	Valid Parameters	ANSICRT	DECCRT	AVO	EDIT	BLOCK[1]
ANSI-Defined Escape Sequences						
HVP	All	x	x			
IND		x	x			
NEL		x	x			
RI		x	x			
RIS		x	x			
SCS	UK,ASCII,0	x				
SCS	UK,ASCII	x	x			
SGR	0,4,7	x	x			
SGR	0,1,4,5,7			x		
DA	Terminal specific			x		
HTS			x			
RM	Class specific		x			
SM	Class specific		x			
TBC	0,3		x			
DCH	All				x	x
DL	All				x	x
IL	All				x	x
Digital-Private Escape Sequences						
DECDHDL	2,3		x			
DECDWL	6		x			
DECKPAM			x			
DECKPNM			x			
DECRC	8		x			
DECSC	7		x			
DECSTBM	All		x			
DECSWL	5		x			
DECPRO	0,1,4,5,7,254					x
DECTTC	0,1					x
DECXMIT	5					x
ANSI Selectable Modes (Set with ANSI SM/RM)						
IRM	4				x	x
GATM	1				x	x
ERM	6					x
TTM	16					x

[1]Terminal characteristics. Prefix is TT2$M_.

(continued on next page)

Table B–2 (Cont.) Sequences and Modes

Name	Valid Parameters	ANSICRT	DECCRT	AVO	EDIT	BLOCK[1]
Digital-Private Selectable Modes (Set with ANSI SM/RM)						
DECCKM	1			x		
DECANM	2			x		
DECCOLM	3			x		
DECSCLM	4			x		
DECSCNM	5			x		
DECOM	6			x		
DECAWM	7			x		
DECARM	8			x		
DECEDM	10					x
DECEKEM	16					x
DECLTM	11					x
DECSCFDM	13					x
DECTEM	14					x
ANSI Assumed Modes						
CRM		Reset	Reset			
EBM		Reset	Reset			
ERM		Set	Set		[2]	
FEAM		Reset	Reset			
FETM		Reset	Reset			
GATM		N/A	N/A		[2]	
HEM		N/A	N/A			
IRM		Reset	Reset	[2]	[2]	
KAM		Reset	Reset			
MATH		N/A	N/A			
PUM		Reset	Reset			
SATM		N/A	N/A			
SRTM		Reset	Reset			
TSM		Reset	Reset			
TTM		N/A	N/A		[2]	
VEM		N/A	N/A			

[1]Terminal characteristics. Prefix is TT2$M_.

[2]Selectable mode.

C
Control Connection Routines

This appendix lists and describes the calling conventions for the pseudoterminal driver control connection routines. The routines appear in this section in alphabetical order. Table C–1 lists the control connection routines and their functions.

Table C–1 Control Connection Routines

Routine Name	Description
PTD$CANCEL	Cancels a queued control connection read request
PTD$CREATE	Creates a pseudoterminal
PTD$DELETE	Deletes a pseudoterminal
PTD$READ	Reads data from the pseudoterminal
PTD$READW	Reads data from the pseudoterminal and waits for read to complete
PTD$SET_EVENT_NOTIFICATION	Enables or disables terminal event notification ASTs
PTD$WRITE	Writes data to the pseudoterminal

PTD$CANCEL—Cancel Queued Request

Cancels a queued control connection read request.

Format

PTD$CANCEL chan

Returns

OpenVMS usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

chan

OpenVMS usage: channel
type: word (unsigned)
access: read only
mechanism: by value

Number of the I/O channel assigned to the pseudoterminal. This channel is only intended to be used for PTD$XXX operations.

Return Values

SS$_NORMAL	Normal successful completion.
SS$_DEVOFFLINE	Device is off line and request cannot proceed.
SS$_IVCHAN	Illegal channel.
SS$_NOPRIV	Insufficient privilege to perform request.

PTD$CREATE—Create a Pseudoterminal

Creates a new pseudoterminal with a unique device name.

Format

PTD$CREATE chan [,acmode] [,charbuff] [,buflen] [,astadr] [,astprm] [,ast_acmode], inadr

Returns

OpenVMS usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

chan

OpenVMS usage: channel
type: word (unsigned)
access: write only
mechanism: by reference

Number of the channel that is assigned to the new pseudoterminal. This argument is the address of a word into which PTD$CREATE writes the channel number. This channel is only intended to be used for PTD$XXX operations.

acmode

OpenVMS usage: access_mode
type: longword (unsigned)
access: read only
mechanism: by value

Access mode to be associated with the channel. The most privileged access mode is the access mode of the caller. I/O operations on the channel can be performed only from equal and more privileged access modes.

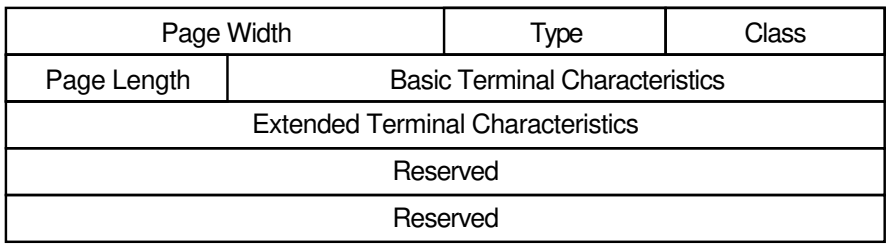
charbuff

OpenVMS usage: device_characteristics
type: longword (unsigned)
access: read only
mechanism: by reference

Address of buffer containing the device characteristics. This information is used to set up the pseudoterminal's initial characteristics. This buffer can be 12, 16, or 20 bytes long.

Figure C–1 shows the format of this buffer.

Figure C–1 Device Characteristics Buffer



ZK–9573–GE

buflen

OpenVMS usage: word_unsigned
type: word (unsigned)
access: read only
mechanism: by value

Length of the characteristics buffer (either 12, 16, or 20 bytes). This argument is required if you supply the **charbuff** argument.

astadr

OpenVMS usage: ast_procedure
type: procedure value
access: call without stack unwinding
mechanism: by reference

AST service routine to be executed when the terminal connection deassigns the last channel to the pseudoterminal. This argument is the procedure value of this routine. This is a repeating AST and is active until the control connection deletes the pseudoterminal.

astprm

OpenVMS usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

AST parameter to be passed to the AST service routine specified by **astadr**.

ast_acmode

OpenVMS usage: access_mode
type: longword (unsigned)
access: read only
mechanism: by value

Access mode for which the AST is to be declared. The most privileged access mode is the access mode of the caller. The resulting mode is the access mode at which the AST is declared.

inadr

OpenVMS usage: address_range
type: longword (unsigned)
access: read only
mechanism: by reference

Address of a two-longword array containing the starting and ending virtual addresses in the virtual address space of the process (either P0 or P1 regions) to be used as I/O buffers. The array contains, in order, the starting and ending virtual addresses. The addresses supplied to *inadr* must express an integral number of CPU-specific pages. The lower address must be on a CPU-specific page boundary, and the higher address must be one less than a CPU-specific page boundary. Together these addresses form a range from lowest to highest bytes. The pages must already exist and must be fully contained in either P0 or P1 space. All pages in the range must:

- Have identical page protection
- Be writable in the mode of the caller
- Be owned by the same access mode
- Be owned in a mode equal to or less privileged than the caller
- Be of the same page type (process or global)

Description

PTD$CREATE creates a new pseudoterminal with a unique device name. This device name is in the form FTA*n*:, where *n* is the unit number.

When a pseudoterminal is created, it inherits the current system terminal default attributes unless you specify an alternate set of characteristics.

Return Values

SS$_NORMAL	Normal successful completion.
SS$_ACCVIO	Unable to read one of the arguments.
SS$_BADPARAM	Bad parameter value.
SS$_EXBYTLM	Insufficient BYTLM to create device or map buffers.
SS$_EXQUOTA	Insufficient quota to create device.
SS$_EXASTLM	Insufficient AST quota for notification AST.
SS$_INSFMEM	Insufficient memory to create device.
SS$_INSFWSL	Insufficient working set limit to map buffers.
SS$_IVSECFLG	Invalid process or global section flags.
SS$_NOPRIV	No privilege for attempted operation.
SS$_PAGOWNVIO	Page owner violation.
SS$_VA_IN_USE	Virtual address already in use.

PTD$DELETE—Delete a Pseudoterminal

Forces the pseudoterminal to be deleted and frees the channel.

Format

PTD$DELETE chan

Returns

OpenVMS usage: longword (unsigned)
type: write only
access: by value

Arguments

chan

OpenVMS usage: channel
type: word (unsigned)
access: read only
mechanism: by value

Number of the I/O channel assigned to the pseudoterminal. This channel is only intended to be used for PTD$XXX operations.

Description

PTD$DELETE forces the pseudoterminal to be deleted and frees the channel assigned to the pseudoterminal. When a pseudoterminal is deleted, any process using the pseudoterminal (except the control program) is disconnected. PTD$DELETE request causes any pending I/O for the control program to be aborted. It deletes any queued event notification ASTs and returns the I/O buffers back to the application. It also causes the pseudoterminal unit control block (UCB) to be deleted once the reference count returns to zero.

Return Values

SS$_NORMAL	Normal successful completion.
SS$_DEVOFFLINE	Device is off line and request cannot proceed.
SS$_IVCHAN	Illegal channel.
SS$_NOPRIV	Insufficient privilege to perform request.

PTD$READ—Read Data from Pseudoterminal

Reads data from the pseudoterminal. The PTD$READ routine completes asynchronously; that is, it returns to the caller without waiting for the data to be read.

For synchronous completion, use the PTD$READW routine. The PTD$READW routine is identical to the PTD$READ routine in every way, except that PTD$READW returns to the caller after the data is read.

Format

```
PTD$READ  [efn], chan [,astadr] [,astprm] readbuf, readbuf_len
```

Returns

OpenVMS usage:	longword (unsigned)
type:	write only
access:	by value

Arguments

efn

OpenVMS usage:	ef_number
type:	longword (unsigned)
access:	read only
mechanism:	by value

Number of the event flag to be set when PTD$READ returns the requested information. If you do not specify this argument, event flag 0 is used. When PTD$READ begins execution, it clears this flag.

chan

OpenVMS usage:	channel
type:	word (unsigned)
access:	read only
mechanism:	by value

Number of the I/O channel assigned to the pseudoterminal. This channel is only intended to be used for PTD$XXX operations.

astadr

OpenVMS usage:	ast_procedure
type:	procedure value
access:	call without stack unwinding
mechanism:	by reference

AST service routine to be executed when PTD$READ completes. If you specify **astadr**, the AST routine executes at the same access mode as the caller of the PTD$READ routine.

astprm

OpenVMS usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

AST parameter to be passed to the AST service routine specified by the **astadr** argument.

readbuf

OpenVMS usage: char_string
type: character coded text string
access: write only
mechanism: by reference

Address of the read I/O status longword. The first character position in an I/O buffer to receive all output is this address plus 4. The **readbuf** argument must be in the range specified in the **inadr** argument of the PTD$CREATE routine, otherwise an SS$_ACCVIO status is returned.

readbuf_len

OpenVMS usage: word_unsigned
type: word (unsigned)
access: read only
mechanism: by value

Number of characters that can be read from the pseudoterminal and stored in the buffer specified by **readbuf**.

Description

The PTD$READ routine reads data from the pseudoterminal. The read request completes with a minimum of one character and a maximum of the number of characters specified by the **readbuf_len** argument. The read operation completes when the pseudoterminal has characters to output. If a read request is issued and no data is available, the read request is queued and then completed at a later time.

Return Values

SS$_NORMAL	Normal successful completion.
SS$_ACCVIO	Unable to read an argument, or invalid read buffer address.
SS$_DEVOFFLINE	Device is off line and request cannot proceed.
SS$_EXASTLM	Insufficient AST quota for notification AST.
SS$_ILLEFC	Illegal event flag cluster.
SS$_INSFMEM	Insufficient memory.
SS$_IVBUFLEN	Buffer size supplied is illegal.
SS$_IVCHAN	Illegal channel.
SS$_NOPRIV	Insufficient privilege to perform request.
SS$_UNASEFC	Unassociated event flag cluster.

PTD$READW—Read Data from Pseudoterminal and Wait

Reads data from the pseudoterminal. The PTD$READW routine completes synchronously, that is, it returns to the caller after the data has been read.

For asynchronous completion, use the PTD$READ routine. The PTD$READ routine is identical to the PTD$READW routine in every way except that PTD$READ returns to the caller without waiting for the data to be read.

Format

```
PTD$READW   [efn], chan [,astadr] [,astprm] readbuf, readbuf_len
```

PTD$SET_EVENT_NOTIFICATION—Enable or Disable Terminal Event Notification ASTs

Enables or disables a number of repeating terminal event notification ASTs.

Format

PTD$SET_EVENT_NOTIFICATION chan, astadr [,astprm] [,acmode], type

Returns

OpenVMS usage: longword (unsigned)
type: write only
access: by value

Arguments

chan

OpenVMS usage: channel
type: word (unsigned)
access: read only
mechanism: by value

Number of the I/O channel assigned to the pseudoterminal. This channel is only intended to be used for PTD$XXX operations.

astadr

OpenVMS usage: ast_procedure
type: procedure value
access: call without stack unwinding
mechanism: by reference

Address of the notification AST service routine, or zero if the AST is to be canceled.

astprm

OpenVMS usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

AST parameter to be passed to the AST service routine specified by the **astadr** argument.

acmode

OpenVMS usage: access_mode
type: longword (unsigned)
access: read only
mechanism: by value

Access mode for which the AST is to be declared. The most privileged access mode is the access mode of the caller. The resulting mode is the access mode at which the AST is declared.

type

OpenVMS usage: type_longword
type: longword (unsigned)
access: read only
mechanism: by value

Value that indicates which notification AST to enable. The $PTDDEF macro defines the symbolic names listed in Table C–2.

Table C–2 Symbolic Names Defined by $PTDDEF Macro

Symbolic Name	Description
PTD$C_SEND_XON	Deliver notification AST when pseudoterminal is ready to accept input. This AST is not delivered if the pseudoterminal is set to NO HOSTSYNC.
PTD$C_SEND_BELL	Deliver notification AST when pseudoterminal wants to stop input and signal it with a bell character.
PTD$C_SEND_XOFF	Deliver notification AST when pseudoterminal wants to stop input and signal it with a DC3 character.
PTD$C_STOP_OUTPUT	Deliver notification AST when pseudoterminal is stopping output.
PTD$C_RESUME_OUTPUT	Deliver notification AST when pseudoterminal is resuming output.
PTD$C_CHAR_CHANGED	Deliver notification AST when pseudoterminal has changed some device characteristic.
PTD$C_ABORT_OUTPUT	Deliver notification AST when pseudoterminal wants to abort output.
PTD$C_START_READ	Deliver notification AST when pseudoterminal is starting an application's read request. This AST is delivered only if read event notification has been enabled.
PTD$C_MIDDLE_READ	Deliver notification AST when pseudoterminal has finished sending an application's read request prompt string. This AST is delivered only if read event notification has been enabled.
PTD$C_END_READ	Deliver notification AST when pseudoterminal has finished an application's read request. This AST is delivered only if read event notification has been enabled.
PTD$C_ENABLE_READ	Enable terminal read event AST delivery. If this code is used, you cannot supply the **astadr** argument.
PTD$C_DISABLE_READ	Disable terminal read event AST delivery. If this code is used, you cannot supply the **astadr** argument.

Description

PTD$SET_EVENT_NOTIFICATION enables or disables the repeating terminal event notification ASTs listed in Table C–2. Once an event notification AST is enabled, it remains in effect until it is disabled or until the device is deleted.

Return Values

SS$_NORMAL	Normal successful completion.
SS$_ACCVIO	Unable to read an argument, or invalid I/O buffer address.
SS$_BADPARAM	An **astadr**, **astprm**, or **acmode** argument was not zero when enabling or disabling read notification.
SS$_DEVOFFLINE	Device is off line and request cannot proceed.
SS$_EXASTLM	Insufficient AST quota for notification AST.
SS$_INSFMEM	Insufficient memory.
SS$_IVCHAN	Illegal channel.
SS$_NOPRIV	Insufficient privilege to perform request.

PTD$WRITE—Write Data to Pseudoterminal

Inputs data to the pseudoterminal and reads any immediately echoed characters.

Format

PTD$WRITE chan [,astadr] [,astprm], wrtbuf, wrtbuf_len [,echobuf] [,echobuf_len]

Returns

OpenVMS usage: longword (unsigned)
type: write only
access: by value

Arguments

chan

OpenVMS usage: channel
type: word (unsigned)
access: read only
mechanism: by value

Number of I/O channel assigned to the pseudoterminal. This channel is only intended to be used for PTD$XXX operations.

astadr

OpenVMS usage: ast_procedure
type: procedure value
access: call without stack unwinding
mechanism: by reference

AST service routine to be executed when PTD$WRITE completes. If **astadr** is specified, the AST routine executes at the same access mode as the caller of the PTD$WRITE routine.

astprm

OpenVMS usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

AST parameter to be passed to the AST service routine specified by the **astadr** argument.

wrtbuf

OpenVMS usage: char_string
type: character coded text string
access: read only
mechanism: by reference

Address of the write I/O status longword. The first character in an I/O buffer to be written is this address plus 4. The **wrtbuf** must be in the range specified by the **inadr** argument of the PTD$CREATE routine; otherwise an SS$_ACCVIO status is returned.

wrtbuf_len

OpenVMS usage: word_unsigned
type: word (unsigned)
access: read only
mechanism: by value

Number of characters to be written to the pseudoterminal. These characters appear as input to the terminal side of the pseudoterminal.

echobuf

OpenVMS usage: char_string
type: character coded text string
access: write only
mechanism: by reference

Address of the echo I/O status longword. The first character position in an I/O buffer to receive all output is this address plus 4. The **echobuf** must be in the range specified by the **inadr** argument of the PTD$CREATE routine; otherwise an SS$_ACCVIO status is returned.

echobuf_len

OpenVMS usage: word_unsigned
type: word (unsigned)
access: read only
mechanism: by value

Number of characters that can be read from the pseudoterminal. If an echo buffer is specified, up to **echobuf_len** characters can be stored in it.

Description

PTD$WRITE inputs data to the pseudoterminal and reads any immediately echoed characters. PTD$WRITE allows you to specify a buffer to receive any output generated by the write; you do not need to issue a separate read request to read this data.

Return Values

SS$_NORMAL	Normal successful completion.
SS$_ACCVIO	Unable to read an argument, or invalid I/O buffer address.
SS$_DATALOST	The terminal driver type-ahead buffer is full and character written was lost.
SS$_DATAOVERUN	The terminal driver type-ahead buffer is getting full; attempts to send more data might result in loss of characters.
SS$_DEVOFFLINE	Device is off line and request cannot proceed.
SS$_EXASTLM	Insufficient AST quota for notification AST.
SS$_INSFMEM	Insufficient memory.
SS$_IVBUFLEN	Buffer size supplied is illegal.
SS$_IVCHAN	Illegal channel.
SS$_NOPRIV	Insufficient privilege to perform request.

Index

A

ACP controls function
  miscellaneous disk, 1–34
ACP functions, 1–2, 1–32
  attributes, 1–15 to 1–18
  disk quotas, 1–35
  IO$_ACCESS, 1–7, 1–9, 1–13, 1–25
  IO$_ACPCONTROL, 1–7
  IO$_CREATE, 1–9, 1–10, 1–13, 1–23
  IO$_DEACCESS, 1–12, 1–13, 1–27
  IO$_DELETE, 1–7, 1–28
  IO$_MODIFY, 1–7, 1–10, 1–12, 1–13, 1–27
  IO$_MOUNT, 1–32
  magnetic tape positioning, 1–33
  quota file transfer block, 1–36
ACP-QIO interface
  See also FIBs
  access file function, 1–25
  access subfunction, 1–9
  ACP control function, 1–32
  ANSI standard, 1–2, 1–34
  arguments, 1–2
    disk quota, 1–36
  attribute control block, 1–14
  attributes, 1–15 to 1–18
  attributes statistics block, 1–20
  BLISS-32 programming, 1–2
  create file function, 1–23
    disk, 1–24
    magnetic tape, 1–25
  deaccess file function, 1–27
  delete file function, 1–28
  description, 1–1
  directory entries, 1–9, 1–25
  FIB (file information block), 1–3
  file characteristics, 1–18
  function codes, A–1
  function modifiers, 1–2
    IO$M_ACCESS, 1–9, 1–23, 1–25
    IO$M_CREATE, 1–23, 1–24, 1–25
    IO$M_DELETE, 1–23, 1–24, 1–29
    IO$M_DMOUNT, 1–32, 1–34
  I/O operations, 1–1
  I/O status block, 1–37
  MACRO programming, 1–1
  movefile subfunction, 1–29
ACP-QIO interface (cont'd)
  record attributes area, 1–18
    values, 1–19
  serious exception (EOT), 1–23, 1–26, 1–34
  status returns, A–1
  XQP (extended QIO processor), 1–1
ACP subfunctions, 1–7
  access, 1–9
  directory lookup, 1–7
  extend, 1–10, 1–37
  movefile, 1–27
  read/write attributes, 1–13
  truncate, 1–12
ALTMODE key, 5–21
ANSI escape sequence, B–9
Applications
  connecting to LAT ports, 5–49
Arguments
  device- or function-dependent, 1–2
  list, A–1
ASCII character set
  See DEC Multinational character set
ASTs (asynchronous system trap)
  quota, 2–28, 3–10, 4–5, 5–44
Attention AST, 5–77
  LAN drivers, 8–47
  mailbox, 4–12
  terminal, 5–44
AUCBs (audio control blocks)
  defining, 2–20
Audio
  error handling in applications, 2–23
  extensions to SCSI disk class driver, 2–19
  $QIO interface to disk class driver, 2–20
  storing with data on CD–ROM, 2–25
Audio applications
  programming, 2–26
  programming example, 2–26
Audio control blocks
  See AUCBs

B

Baud rate
  of terminal, 5–42

64-bit virtual addressing
device driver support, xvii
BOT (beginning-of-tape)
See Magnetic tapes, BOT marker
Broadcast messages, 5–19, 5–22, 5–24, 5–49
Buffered I/O quota, 2–28, 3–10, 4–5

C

Caches
tape, 3–5
write-back volatile, 3–5
Carriage control, 5–38
Characters
terminator set for, 5–30
Character sets
See DEC Multinational character set
terminal lowercase, 5–21
Compact discs
See CD–ROM (compact disc read-only memory)
Configuring ISA devices, 8–9 to 8–12
CONNECT command, 5–18
Console disks
See RX01 console disk
Console terminals, 5–1
Control characters
list, B–1
terminal, 5–4 to 5–7, 5–10
Control connection routines, C–1 to C–14
PTD$CANCEL, C–2
PTD$CREATE, C–3
PTD$DELETE, C–6
PTD$READ, C–7
PTD$READW, C–9
PTD$SET_EVENT_NOTIFICATION, C–10
PTD$WRITE, C–13
Control key sequences, 5–8
Create file function, 1–23
directory entry creation, 1–25
CTDRIVER driver, 5–12, 5–37
Ctrl/x key sequence
See Terminals, control characters

D

Data
storing with audio information on CD–ROM, 2–25
Data checks
disk, 2–12, 2–33, 2–34
magnetic tape, 3–6, 3–14, 3–15
Deaccess file function, 1–27
DEC Multinational character set, B–1
Delete file function, 1–28
Delete key, 5–5
Device characteristics
disk, 2–26
LAN drivers, 8–26
magnetic tape, 3–8
mailbox, 4–4
pseudoterminal, 6–3
terminal, 5–20
Devices
SCSI support, 7–2
supported, 7–2
DHU11 device, 5–1
DHV11 device, 5–1
Dialup lines, 5–13
Digital private escape sequence, B–9
Direct I/O count process limit, 2–28, 3–10
Directory entry
creation, 1–25
protection, 1–9
Directory lookup subfunction, 1–7
directory entry protection, 1–9
DISCONNECT command, 5–18
Disk class drivers
audio extensions, 2–19
$QIO interface to audio functions, 2–20
Disk drives
compatibility for volume shadowing, 7–2
Disks
See also DSA disks
ACP operation
control function, 1–34
creating file, 1–24
deaccessing file, 1–27
audio functions
See SCSI class driver
available function, 2–36
Backup utility, 2–17
compact disc, 2–5
data check, 2–12, 2–33, 2–34
device characteristics, 2–26
device descriptions, 2–3 to 2–7
driver, 2–1, 2–19
SCSI, 2–18
VAXstation 2000 and MicroVAX 2000, 2–18
dual-pathed, 2–8
DSA disks, 2–11
dual-ported, 2–9
dual porting
restrictions for use, 2–10
dual-porting
DSA disks, 2–11
HSC disks, 2–11
error recovery, 2–14
features, 2–8
file attributes, 2–12
function codes, 2–29, A–2
function modifiers
IO$M_DATACHECK, 2–12, 2–33, 2–34

Disks
function modifiers (cont'd)
IO$M_DELDATA, 2–34
IO$M_ERASE, 2–31, 2–34
IO$M_INHRETRY, 2–14, 2–33, 2–34
HSC50 controller, 2–2
HSC70 controller, 2–2
HSC controllers, 2–2
I/O functions, 2–28, 2–33 to 2–39
See also ACP-QIO interface
arguments, 2–30 to 2–33
IO$_ACPCONTROL, 1–34
I/O status block, 2–39
KDA50 controller, 2–2
KDB50 controller, 2–2
KFQSA adapter, 2–3
offset recovery, 2–12
pack acknowledge function, 2–36
port access mode, 2–9
port selection, 2–9
programming example, 2–40
quotas, 1–35 to 1–36, 2–28
RA60, 2–3
RA70, 2–3
RA90, 2–3
RB02 cartridge, 2–4
RC25, 2–4
RCT (replacement and caching table), 2–17
RD53, 2–4
RD54, 2–4
read function, 2–33
RF30, 2–5
RF31
failover, 2–11
RF70
failover, 2–11
RF71, 2–5
RK06 cartridge, 2–5
RK07 cartridge, 2–5
RL02 cartridge, 2–4
RM03, 2–5
RM05, 2–5
RP05, 2–5
RP06, 2–5
RP07, 2–5
RQDX3 controller, 2–3
RRD40 CD–ROM, 2–5
RRD50 CD–ROM, 2–5
RX02, 2–6
RX23 flexible, 2–7
RX33 flexible, 2–7
RX50 flexible, 2–7
RZ22, 2–7
RZ23, 2–7
RZ55, 2–7
SDI (Standard Disk Interface), 2–3
search function, 2–35
sector translation, 2–15
Disks (cont'd)
seek operations, 2–13, 2–36
sense mode function, 2–35
set density function, 2–35
set preferred path function, 2–37
SII integral adapter, 2–3
skip sectoring, 2–14
status returns, A–3
supported devices, 2–1 to 2–8
SYS$GETDVI returns, 2–26
TU58 magnetic tape, 2–7
data checks, 2–12
read function, 2–33
search function, 2–35
write check function, 2–37
write function, 2–34
UDA50 disk adapter, 2–1
unload function, 2–36
use with Verify utility, 2–16, 2–17
VAXstation 2000 and MicroVAX 2000 driver, 2–18
write check function, 2–36
write function, 2–34
DISMOUNT command, 1–34
DMB32 device, 5–1
DMF32 device, 5–1
DMZ32 device, 5–1
Drivers
disks, 2–1
LAN, 8–2
LAT port, 5–1
magnetic tape, 3–1
mailbox, 4–1
pseudoterminal, 6–1
SCSI, 2–18
SCSI disk class, 2–19
shadow set virtual unit, 7–1
terminal, 5–1
VAXstation 2000 and MicroVAX 2000 disk, 2–18
DSA32 device, 5–1
DSA disks, 2–11, 2–16
See also Disks
bad block replacement, 2–17, 2–18
forced error, 2–17
forced error flag, 2–17
use with Verify utility, 2–16, 2–17
DSE (data security erase)
magnetic tape, 3–22
Dual host
definition, 2–3
Dual path
definition, 2–8
Dual-pathed disks, 2–8
DSA disk, 2–11
Dual-ported disks, 2–9
DSA disk, 2–11
HSC disk, 2–11

Dual-ported disks (cont'd)
restrictions for use, 2–10
Duplex mode
See also Half-duplex mode
terminal, 5–10
DZ11 device, 5–1
DZ32 device, 5–1
DZV11 device, 5–1

E

Enable attention AST function
LAN drivers, 8–47
End-of-file
See EOF
End-of-tape markers
See EOT markers
End-of-volume
See EOV
EOF (end-of-file)
status
magnetic tape, 3–14
write mailbox message, 4–11
EOT markers
status
magnetic tape, 3–14, 3–15, 3–17
EOV (end-of-volume)
detection on magnetic tape, 3–17
Error recovery
disk, 2–14
magnetic tape, 3–6
shadow set virtual unit driver, 7–8
Escape sequences
ANSI, B–9
Digital-private, B–9
terminal, 5–7, 5–21
Ethernet
device drivers, 8–2
Event notification
pseudoterminal, 6–6
Extend subfunction, 1–10

F

FDDI (Fiber Distributed Data Interface)
device drivers, 8–2
FIBs (file information blocks), 1–3
See also ACP function
access control, 1–9
contents, 1–5 to 1–6
descriptor, 1–2, 1–3
directory lookup, 1–7
disk quota, 1–35 to 1–36
extend control, 1–11
format, 1–4
IO$_ACCESS, 1–26
IO$_ACPCONTROL, 1–33 to 1–36
IO$_CREATE, 1–23
FIBs (file information blocks) (cont'd)
IO$_DEACCESS, 1–27
IO$_DELETE, 1–29
IO$_MODIFY, 1–28
truncate control, 1–12
File characteristics
ACP-QIO attributes, 1–18
Form feeds
terminal, 5–22
Full-duplex mode, 5–10
Function codes
See also I/O functions
IO$_ACCESS, 1–25
IO$_ACPCONTROL, 1–32, 3–12
IO$_ADDSHAD, 7–4
IO$_AVAILABLE, 2–36, 3–22, 7–6
IO$_COPYSHAD, 7–5
IO$_CREATE, 1–23
IO$_CRESHAD, 7–3
IO$_DEACCESS, 1–27
IO$_DELETE, 1–28
IO$_DSE, 3–22
IO$_FORMAT, 2–35
IO$_MODIFY, 1–27
IO$_PACKACK, 2–36
IO$_READLBLK, 2–33, 3–13, 4–5, 5–28, 8–29
IO$_READPBLK, 2–33, 3–13, 4–5, 8–29
IO$_READPROMPT, 5–28
IO$_READVBLK, 2–33, 3–13, 4–5, 5–28, 8–29
IO$_REMSHAD, 7–5
IO$_REWIND, 3–16
IO$_REWINDOFF, 3–18
IO$_SEARCH, 2–35
IO$_SEEK, 2–36
IO$_SENSECHAR, 2–35, 5–78, 7–6
IO$_SENSEMODE, 2–35, 3–18, 5–78, 8–52
IO$_SETCHAR, 3–19, 5–40, 8–35
IO$_SETMODE, 3–19, 5–40, 8–35
IO$_SETPRFPATH, 2–37
IO$_SKIPFILE, 3–16
IO$_SKIPRECORD, 3–16
IO$_UNLOAD, 2–36, 3–18
IO$_WRITECHECK, 2–36
IO$_WRITELBLK, 2–34, 3–15, 4–9, 5–37, 8–32
IO$_WRITEOF, 3–17
IO$_WRITEPBLK, 2–34, 3–15, 4–9, 5–37, 8–32
IO$_WRITEVBLK, 2–34, 3–15, 4–9, 5–37, 8–32
list of, A–1
Function modifiers
for LAN driver, 8–32
IO$M_ACCESS, 1–23, 1–25, 3–10
IO$M_ATTNAST, 8–47
IO$M_BRDCST, 5–49, 5–81
IO$M_BREAKTHRU, 5–11, 5–38
IO$M_CANCTRLO, 5–5, 5–38
IO$M_CREATE, 1–23, 1–25, 3–10
IO$M_CTRL, 8–47, 8–52
IO$M_CTRLCAST, 5–44

Function modifiers (cont'd)
IO$M_CTRLYAST, 5–6, 5–44
IO$M_CVTLOW, 5–29
IO$M_DATACHECK, 2–12, 2–33, 2–34, 3–6, 3–14, 3–15
IO$M_DELDATA, 2–34
IO$M_DELETE, 1–23, 1–29
IO$M_DMOUNT, 1–32
IO$M_DSABLMBX, 5–29
IO$M_ENABLMBX, 5–38
IO$M_ERASE, 2–31, 2–34, 3–15
IO$M_ESCAPE, 5–7, 5–29
IO$M_EXTEND, 5–29, 5–31
IO$M_HANGUP, 5–44
IO$M_INCLUDE, 5–45, 5–48
IO$M_INHEXTGAP, 3–7
IO$M_INHRETRY, 2–33, 3–7
IO$M_MAINT, 5–45, 5–47
IO$M_NOECHO, 5–11, 5–26, 5–29
IO$M_NOFILTR, 5–29
IO$M_NOFORMAT, 5–11, 5–38
IO$M_NORSWAIT, 4–9, 4–11
IO$M_NOW, 4–5, 4–9, 4–11, 8–32
IO$M_NOWAIT, 3–16, 3–18
IO$M_OUTBAND, 5–48
IO$M_PURGE, 5–29
IO$M_RD_MODEM, 5–80
IO$M_READATTN, 4–11
IO$M_READERCHECK, 4–9, 4–11, 4–15
IO$M_REFRESH, 5–38
IO$M_RESPONSE, 8–34
IO$M_REVERSE, 3–14
IO$M_SETPROT, 4–14
IO$M_SET_MODEM, 5–45
IO$M_SHUTDOWN, 8–47
IO$M_STREAM, 4–5
IO$M_TIMED, 5–29
IO$M_TRMNOECHO, 5–30
IO$M_TT_ABORT, 5–48
IO$M_TYPEAHDCNT, 5–79
IO$M_UNLOOP, 5–48
IO$M_WRITERCHECK, 4–5, 4–15
list of, A–1

H

Half-duplex mode, 5–10, 5–21
See also Duplex mode
Hangup function modifier
terminal
disconnecting a, 5–44
interaction with, 5–18, 5–25
Hardware
supported, 7–2
HSC40 disk controller, 2–2
HSC50 disk controller, 2–2
HSC70 disk controller, 2–2
HSC disk controllers, 2–2
HSC disks, 2–11

I

I/O buffers
pseudoterminal, 6–3
I/O drivers
LAN drivers, 8–2
I/O functions
See also Function code
ACP-QIO interface, 1–2
codes, A–1
disk, 1–2, 2–28
for LAN driver, 8–28
list of, A–1
magnetic tape, 1–2, 3–9
terminal, 5–27
I/O write operations
preventing data loss, 2–13
unsuccessful completion, 2–13
INITIALIZE command, 3–22
IO$M_LT_QUE_CHG_NOTIF
LAT $QIO Function Modifier, 5–77
IO$M_ROUTE functional modifier, 8–51
IO$M_SENSE_MAC
Used with IO$_SENSEMODE, 8–54
IO$M_SENSE_MAC functional modifier, 8–54
IO$M_SHOW_MAP
Used with IO$_SENSEMODE, 8–56
IO$M_SHOW_MAP functional modifier, 8–56
IO$M_SHOW_ROUTE
Used with IO$_SENSEMODE, 8–56
IO$M_SHOW_ROUTE functional modifier, 8–56
IO$M_UPDATE_MAP functional modifier, 8–49
IO$_M_SET_MAC functional modifier, 8–48
IO$_SETMODE
IO$M_UPDATE_MAP, 8–49
IOSBs (I/O status blocks)
disk, 2–39
LAN drivers, 8–58
LAT port driver, 5–81
magnetic tape, 3–23
mailbox, 4–15
terminal, 5–81
IOSBs (/O status blocks)
ACP-QIO interface, 1–37
Itemlist read operations, 5–31

K

KDA50 disk controller, 2–2
KDB50 disk controller, 2–2
Keyboard control character, 5–4 to 5–7, 5–10
KFQSA adapter, 2–3

L

LAN drivers
address
destination, 8–29, 8–33
Ethernet, 8–6 to 8–7
hardware, 8–53
loopback assistance, 8–7
multicast, 8–6, 8–7, 8–29, 8–40, 8–41
node, 8–6
physical, 8–6, 8–7, 8–29, 8–41, 8–53
port, 8–41
shared protocol destination, 8–38
source, 8–29
Token Ring, 8–8
AST access mode, 8–47
AST service routine address, 8–47
attention AST, 8–47
buffer
receive, 8–29, 8–37
channel assignment, 8–4
characteristics
device, 8–26, 8–52
extended, 8–36 to 8–45, 8–53
controller mode, 8–37
CRC generation, 8–38
device characteristics, 8–26, 8–52
See also LAN, extended characteristics
drivers
initializing, 8–4
operating, 8–4
driver service (802 format), 8–45
echo mode (DEUNA only), 8–38
error summary bits, 8–28
Ethernet, 8–2, 8–4, 8–6, 8–20
Ethernet packet format, 8–20
Ethernet packet padding, 8–21
exclusive mode on Alpha systems, 8–25
exclusive mode on VAX systems, 8–21
extended characteristics, 8–36 to 8–45, 8–52
FDDI, 8–2
function codes, 8–28, A–8
function modifiers, 8–32, 8–34, 8–47 to 8–52
hardware interface, 8–4
I/O functions, 8–29, 8–32, 8–35, 8–52
I/O status block, 8–58
IEEE 802 packet format
Class I service packet format, 8–22, 8–39
driver service parameter, 8–45
extended packet format, 8–25, 8–39
802 format SAP parameter, 8–44
group SAP parameter, 8–40
read function, 8–29
SAP use and restrictions, 8–24
support for, 8–9
user-supplied service packet format, 8–24, 8–39
LAN drivers
IEEE 802 packet format (cont'd)
write function, 8–32
IEEE 802 programming example, 8–60
internal loopback mode (DELUA only), 8–40
loopback mode, 8–37
message size, 8–27, 8–31, 8–32, 8–34, 8–37
modify characteristics, 8–35
multicast address state, 8–41
packet format, 8–19
Class I service, 8–22
Ethernet, 8–20
extended 802, 8–25
IEEE 802, 8–22
set mode parameters, 8–45
SNAP SAP value, 8–25
user-supplied service, 8–24
padding
message size, 8–27, 8–32
transmit messages, 8–41
parameter ID, 8–35
packet format, 8–45
parameter validation, 8–46
port, 8–2
address, 8–36
start, 8–35
privilege, 8–29
programming example, 8–60
programming notes, 8–59
promiscuous mode, 8–42, 8–59
rules for, 8–59
protocol type, 8–2, 8–29, 8–33, 8–43
access mode, 8–36
cross-company, 8–20
Digital, 8–20
Ethernet, 8–20
protocol type on Alpha systems
sharing, 8–25
protocol type on VAX systems
sharing, 8–21
read function, 8–29
restart, 8–43
sense mode function, 8–52
Service Access Point (SAP), 8–24
set controller mode
extended characteristics, 8–36 to 8–45
P2 buffer, 8–35
parameter ID, 8–35
set controller mode on Alpha systems
protocol type sharing, 8–25
set controller mode on VAX systems
protocol type sharing, 8–21
set mode function, 8–35
shared default mode on Alpha systems, 8–25
shared default mode on VAX systems, 8–21
shared with destination mode on Alpha systems, 8–25

LAN drivers (cont'd)
shared with destination mode on VAX systems, 8–21
shutdown controller mode, 8–47
shutdown port, 8–47
software interface, 8–4
status returns, A–8
supported devices, 8–2
SYS$ASSIGN routine, 8–4
SYS$DASSGN routine, 8–5
SYS$GETDVI routine, 8–26
Token Ring, 8–2
transmit/receive buffer size, 8–36
unit and line status, 8–28
write function, 8–32
LAT $QIO, 5–77
LAT port driver (LTDRIVER), 5–1
LAT SENSEMODE $QIO Function, 5–59
LAT SETMODE $QIO Function, 5–51
Line terminator
terminal, 5–10

M

Magnetic tapes
function codes, A–3
I/O functions
See also ACP-QIO interface
status returns, A–4
TU58 magnetic tape
See Disk, TU58
Mailboxes
See also Terminals
creating, 4–1
deleting, 4–3
device characteristics, 4–4
disable terminal, 5–22
driver, 4–1
function codes, 4–5, A–5
function modifiers
IO$M_NORSWAIT, 4–9, 4–11
IO$M_NOW, 4–3, 4–5, 4–9, 4–11, 4–13
IO$M_READATTN, 4–11
IO$M_READERCHECK, 4–9, 4–11, 4–15
IO$M_SETPROT, 4–14
IO$M_STREAM, 4–5
IO$M_WRITERCHECK, 4–5, 4–15
get mailbox information function, 4–15
I/O functions
IO$_READLBLK, 4–5
IO$_READPBLK, 4–5
IO$_READVBLK, 4–5
IO$_WRITELBLK, 4–9
IO$_WRITEOF, 4–11
IO$_WRITEPBLK, 4–9
IO$_WRITEVBLK, 4–9
I/O status block, 4–15
list of operations, 4–1
Mailboxes (cont'd)
message format, 4–4
terminal, 5–19
message size, 4–1
multiport memory, 4–1
permanent, 4–1, 4–3, 4–4
programming examples, 4–17
protection, 4–1, 4–4, 4–14
read attention AST function, 4–11
read function, 4–5
set attention AST function, 4–11
set protection function, 4–14
status returns, A–5
SYS$GETDVI returns, 4–4
temporary, 4–1, 4–4
terminal/mailbox interaction, 5–18
volume protection, 4–14, 4–15
wait for writer/reader function, 4–14
write attention AST function, 4–11
write end-of-file message function, 4–11
write function, 4–9
Master adapter, 3–5
Message format
See Mailboxes
Modify file function, 1–27
MOUNT command, 3–22
Mount function, 1–32
Movefile subfunction
calling, 1–18, 1–29
description, 1–29
MSCP (mass storage control protocol)
supported devices, 7–2
Multinational character set
See DEC Multinational character set
Multiplexers, 5–1
DMB32 device, 5–1
DMF32 device, 5–1
DZ11 device, 5–1
DZ32 device, 5–1

O

Out-of-band AST, 5–14, 5–48

P

Parity flag, 5–42
PASTHRU mode, 5–9, 5–11, 5–26, 5–28
Permanent mailboxes
See Mailboxes
Port access mode, 2–9
Port selection, 2–9
Protection
See also Mailboxes
directory entry, 1–9

Pseudoterminals
canceling request, 6–2
control connection routines, C–1
creating, 6–1
deleting, 6–2
device characteristics, 6–3
driver, 6–1
event notification, 6–6
features, 6–3
flow control, 6–5
I/O buffers, 6–3
programming example, 6–7
reading data, 6–5
using write with echo, 6–5
writing data, 6–5
PTD$CANCEL control connection routine, C–2
PTD$CREATE control connection routine, C–3
PTD$DELETE control connection routine, C–6
PTD$READ control connection routine, C–7
PTD$READW control connection routine, C–9
PTD$SET_EVENT_NOTIFICATION control connection routine, C–10
PTD$WRITE control connection routine, C–13

Q

$QIO, 5–77
Quota file transfer block, 1–36
Quotas
AST, 2–28, 3–10, 4–5, 4–12, 5–44
buffered I/O, 2–28, 3–10, 4–5
BYTELIM, 1–10
direct I/O, 2–28, 3–10
disk, 1–35 to 1–36
mailbox buffer, 4–1, 4–3, 4–5

R

RA60 disk, 2–3
RA70 disk, 2–3
RA90 disk, 2–3
RB02 cartridge disk, 2–4
RC25 disk, 2–4
RD53 disk, 2–4
RD54 disk, 2–4
Read attention AST function, 4–12
Read/write attributes subfunction, 1–13
Record attributes value, 1–19
Return key, 5–7
Rewind offline function, 3–18
RF30 disk, 2–5
RF71 disk, 2–5
RK06 cartridge disk, 2–5
RK07 cartridge disk, 2–5
RL02 cartridge disk, 2–4
RM03 disk, 2–5
RM05 disk, 2–5
RP05 disk, 2–5
RP06 disk, 2–5
RP07 disk, 2–5
RQDX3 disk controller, 2–3
RTPAD component of SET HOST, 5–12
RX01 console disk, 2–6
RX02 diskette, 2–6
RX23 diskette, 2–7
RX33 diskette, 2–7
RX50 diskette, 2–7
RZ22 disk, 2–7
RZ23 disk, 2–7
RZ55 disk, 2–7

S

SCSI (Digital Small Computer Systems Interface)
hardware compliance, 7–2
SCSI (Small Computer Systems Interface)
disk class driver, 2–19
disks
class driver, 2–18
error recovery, 2–14, 2–18
handling errors in audio applications, 2–23
$QIO interface to disk class driver, 2–20
Sector translation, 2–15
Seek operation, 2–13
Sense tape mode function, 3–18
Serial line multiplexer, 5–1
Set attention AST
See Attention AST
SET HOST facility, 5–11
Set Mode
Set Mac qualifier, 8–48
Set modes
magnetic tape, 3–19
mailbox, 4–11
terminal, 5–40
SET TERMINAL command, 5–4, 5–19, 5–26
Setting characteristics
magnetic tape, 3–19
terminal, 5–40
Shadow set virtual unit driver, 7–1
functions, 7–2
hardware configurations, 7–2
SHDRIVER.EXE file, 7–1
Shelving
determining if file is shelvable, 1–18
determining if file is shelved, 1–18
SII integral adapter, 2–3
Skip file function, 3–16
Skip sectoring, 2–14
Slave formatter, 3–5
Small Computer Systems Interface (SCSI)
See SCSI

SS$_ABORT return, 5–47, 5–75, 8–43, A–3, A–4, A–5, A–7, A–8
SS$_ACCONFLICT return, A–1
SS$_ACCVIO return, 4–15, A–5, A–8
SS$_ACPVAFUL return, A–1
SS$_BADATTRIB return, A–1
SS$_BADCHKSUM return, A–1
SS$_BADESCAPE return, 5–8, A–7
SS$_BADFILEHDR return, A–1
SS$_BADFILENAME return, A–1
SS$_BADFILEVER return, A–1
SS$_BADIRECTORY return, A–1
SS$_BADPARAM return, 8–21, 8–25, 8–36, 8–46, A–1, A–7, A–8
SS$_BADQFILE return, A–1
SS$_BLOCKCNTERR return, A–1
SS$_BUFFEROVF return, 4–5, 8–53, A–5, A–8
SS$_CANCEL return, A–3, A–4, A–5, A–7
SS$_CHANINTLK return, A–7
SS$_COMMHARD return, A–8
SS$_CONTROLC return, 5–48, A–7
SS$_CONTROLO return, A–7
SS$_CONTROLY return, A–7
SS$_CREATED return, A–1
SS$_CTRLERR return, A–3, A–4, A–8
SS$_DATACHECK return, A–3, A–4, A–8
SS$_DATAOVERUN return, 5–9, 8–32, A–3, A–4, A–7, A–8
SS$_DEVACTIVE, 5–77
SS$_DEVACTIVE return, 5–75, A–8
SS$_DEVALLOC return, A–8
SS$_DEVICEFULL return, A–1
SS$_DEVINACT return, A–8
SS$_DEVOFFLINE return, A–4, A–8
SS$_DEVREQERR, 5–77
SS$_DEVREQERR return, A–8
SS$_DIRFULL return, A–1
SS$_DIRNOTEMPTY return, A–1
SS$_DISCONNECT return, A–8
SS$_DRVERR return, A–3, A–4
SS$_DUPDSKQUOTA return, A–1
SS$_DUPFILENAME return, A–1
SS$_DUPUNIT return, A–8
SS$_ENDOFFILE return, 3–17, 4–8, 4–11, 8–32, A–1
  LAN driver status return, A–8
  magnetic tape status return, A–4
  mailbox status return, A–5
SS$_ENDOFTAPE return, A–4
SS$_ENDOFVOLUME return, 3–17, A–4
SS$_EXBYTLM return, A–1
SS$_EXDISKQUOTA return, A–1
SS$_EXQUOTA return, 4–15, A–5, A–8
SS$_FCPREADERR return, A–1
SS$_FCPREWNDERR return, A–1
SS$_FCPSPACERR return, A–1
SS$_FCPWRITERR return, A–1
SS$_FILELOCKED return, A–1
SS$_FILENUMCHK return, A–1
SS$_FILEPURGED return, A–1
SS$_FILESEQCHK return, A–1
SS$_FILESTRUCT return, A–1
SS$_FILNOTEXP return, A–1
SS$_FORCEDERR return, A–3
SS$_FORMAT return, A–3, A–4
SS$_HANGUP return, 5–14
SS$_HEADERFULL return, A–1
SS$_IBCERROR return, A–1
SS$_IDXFILEFULL return, A–1
SS$_ILLCNTRFUNC return, A–2
SS$_ILLIOFUNC return, 4–15, 5–75, A–3, A–4, A–5
SS$_INCOMPAT return, A–7
SS$_INSFMAPREG return, A–8
SS$_INSFMEM return, 4–15, A–5, A–8
SS$_IVADDR return, A–3
SS$_IVBUFLEN return, 8–34, A–3, A–8
SS$_MBFULL return, 4–2, 4–9, 4–15, A–5
SS$_MBTOOSML return, 4–15, A–5
SS$_MEDOFL return, A–3, A–4, A–8
SS$_NODISKQUOTA return, A–2
SS$_NOMOREFILES return, A–2
SS$_NONEXDRV return, A–3, A–4
SS$_NOPRIV return, 4–14, 4–15, A–2, A–5, A–8
SS$_NOQFILE return, A–2
SS$_NOREADER return, A–5
SS$_NORMAL return, 4–15, 5–75, A–3, A–4, A–7, A–8
SS$_NOSUCHFILE return, A–2
SS$_NOTAPEOP return, A–2
SS$_NOTLABELMT return, A–2
SS$_NOTPRINTED return, A–2
SS$_NOTVOLSET return, A–2
SS$_NOWRITER return, A–5
SS$_OPINCOMPL return, 8–43, A–3, A–4, A–8
SS$_OVRDSKQUOTA return, A–2
SS$_PARITY return, A–3, A–4, A–7
SS$_PARTESCAPE return, 5–8, 5–32, A–7
SS$_QFACTIVE return, A–2
SS$_QFNOTACT return, A–2
SS$_RCT return, A–3
SS$_RDDELDATA return, A–3
SS$_SERIOUSEXCP return, A–2, A–4
SS$_SUPERSEDE return, A–2
SS$_TAPEPOSLOST return, A–2
SS$_TIMEOUT return, 5–29, 5–75, 8–43, A–3, A–4, A–7, A–8
SS$_TOOMANYVER return, A–2
SS$_TOOMUCHDATA return, A–8
SS$_UNSAFE return, A–3, A–4
SS$_VOLINV return, A–3, A–4
SS$_WASECC return, A–3

SS$_WRITLCK return, A–2, A–3, A–4
SS$_WRONGACP return, A–2
Standard Disk Interface (SDI), 2–3
SYS$ASSIGN routine, 4–1, 5–18, 5–76, 8–4
SYS$CREMBX system service, 4–1
SYS$DASSGN routine, 4–3, 8–4
SYS$DELMBX system service, 4–3
SYS$DISMOU system service, 1–34
SYS$GETDVI routine, 3–8
  disk, 2–26
  LAN drivers, 8–26
  mailbox, 4–4
  terminal, 5–20
SYS$QIO system service
  interface to audio functions, 2–20
System console terminal, 5–1

T

Tabs
  Ctrl/I, 5–7
  terminal mechanical, 5–22
  terminal tab stops, 5–37
Tape marks, 3–14, 3–16, 3–17
Tapes
  function modifiers
    IO$M_DATACHECK, 3–6, 3–14
    IO$M_INHEXTGAP, 3–7
    IO$M_INHRETRY, 3–7
    IO$M_REVERSE, 3–14
  I/O functions
    IO$_ACCESS, 3–10
    IO$_ACPCONTROL, 1–33, 3–12
    IO$_AVAILABLE, 3–22
    IO$_CREATE, 3–10
    IO$_DEACCESS, 3–10
    IO$_DSE, 3–10, 3–22
    IO$_FLUSH, 3–10
    IO$_MODIFY, 3–10, 3–22
    IO$_PACKACK, 3–22
    IO$_READLBLK, 3–13
    IO$_READPBLK, 3–13
    IO$_READVBLK, 3–13
    IO$_REWIND, 3–16
    IO$_REWINDOFF, 3–18
    IO$_SENSEMODE, 3–18
    IO$_SETCHAR, 3–19
    IO$_SETMODE, 3–19
    IO$_SKIPFILE, 3–16
    IO$_SKIPRECORD, 3–16
    IO$_UNLOAD, 3–18
    IO$_WRITELBLK, 3–15
    IO$_WRITEOF, 3–17
    IO$_WRITEPBLK, 3–15
    IO$_WRITEVBLK, 3–15
  modify function, 3–22
  sense mode function, 3–18
Temporary mailboxes, 4–4
Terminal characteristics, 5–21 to 5–26
Terminals
  ANSI CRT terminal, 5–23
  autobaud detection, 5–19, 5–23
  baud rate, 5–19, 5–23, 5–42
  bell (Ctrl/G), 5–9
  broadcast messages, 5–19, 5–22, 5–24, 5–49
  carriage control, 5–38
  command line editing, 5–3, 5–37
  command recall (Ctrl/B), 5–4, 5–6
  control and data signals, 5–16
  control characters, 5–4 to 5–7, 5–10, 5–28
    numeric values, B–1
  control key sequences, 5–8
  cursor movement, 5–3, 5–6, 5–23
  delete character, 5–3
  delete line (Ctrl/U), 5–5, 5–28
  device characteristics, 5–20, 5–21
    categories, 5–27
    changing, 5–43
    extended, 5–23
  dial-up
    characteristic, 5–22
    lines, 5–13, 5–25
    support, 5–13
  dialup
    lines, 5–44
  Digital CRT terminal, 5–24, 5–25
  discard output (Ctrl/O), 5–5, 5–28, 5–38
  driver, 5–1
  duplex modes, 5–10, 5–13
  enable Ctrl/C AST, 5–44
  enable Ctrl/Y AST, 5–44
  escape sequences, 5–7, 5–81
    ANSI, B–9
    Digital-private, B–9
    overflow size (item code), 5–32
  extended characteristics, 5–23
  fallback conversion, 5–11, 5–25, 5–43
  features, 5–2
  form feed, 5–22, 5–37
  frame size, 5–43
  function codes, 5–27, A–6
  function modifiers
    See also Terminals
    IO$M_BRDCST, 5–49, 5–81
    IO$M_BREAKTHRU, 5–11, 5–38
    IO$M_CANCTRLO, 5–5, 5–38
    IO$M_CTRLCAST, 5–44
    IO$M_CTRLYAST, 5–6, 5–14, 5–44
    IO$M_CVTLOW, 5–29
    IO$M_DSABLMBX, 5–29
    IO$M_ENABLMBX, 5–38
    IO$M_ESCAPE, 5–7, 5–29
    IO$M_EXTEND, 5–29, 5–31
    IO$M_HANGUP, 5–44
    IO$M_INCLUDE, 5–20, 5–45, 5–48

Terminals
function modifiers (cont'd)
IO$M_LOOP, 5–47
IO$M_LT_CONNECT, 5–51
IO$M_LT_DISCON, 5–51
IO$M_LT_SENSEMODE, 5–51
IO$M_LT_SETMODE, 5–51
IO$M_MAINT, 5–45, 5–47
IO$M_NOECHO, 5–9, 5–11, 5–26, 5–29
IO$M_NOFILTR, 5–29
IO$M_NOFORMAT, 5–11, 5–38, 5–47
IO$M_OUTBAND, 5–48
IO$M_PURGE, 5–29
IO$M_RD_MODEM, 5–80
IO$M_REFRESH, 5–38
IO$M_SET_MODEM, 5–45
IO$M_TIMED, 5–29
IO$M_TRMNOECHO, 5–30
IO$M_TT_ABORT, 5–20, 5–48
IO$M_TYPEAHDCNT, 5–79
IO$M_UNLOOP, 5–48
hangup, 5–14, 5–17, 5–18, 5–25, 5–44, 5–78
I/O functions
CTDRIVER, 5–37
IO$_READLBLK, 5–28
IO$_READPROMPT, 5–28
IO$_READVBLK, 5–28
IO$_SENSECHAR, 5–78
IO$_SENSEMODE, 5–78
IO$_SETCHAR, 5–40
IO$_SETMODE, 5–40
IO$_TTY_PORT, 5–51
IO$_WRITELBLK, 5–37
IO$_WRITEPBLK, 5–37
IO$_WRITEVBLK, 5–37
I/O status block, 5–81
initiate login, 5–10
input processing, 5–3
insert/overstrike (Ctrl/A), 5–3, 5–7
interrupt (Ctrl/Y), 5–6
item codes, 5–32 to 5–36
itemlist read, 5–31
item codes, 5–32 to 5–36
item descriptor, 5–31
LAT line, 5–1
LAT port driver, 5–49
application services creation, 5–76
I/O functions, 5–51
LAT rejection codes, 5–84
line editing, 5–3, 5–25
See also Terminal, item codes
line feed, 5–37
line terminators, 5–10
mailbox, 5–18, 5–38
message format, 5–19
message types, 5–19
modem
characteristic, 5–22
Terminals
modem (cont'd)
control signals, 5–16
data signals, 5–16
protocol, 5–14
sense signals, 5–80
signal control, 5–13
no type-ahead, 5–22
out-of-band
See also Out-of-band AST
characters, 5–20
output
CTDRIVER, 5–12
RTPAD, 5–12
SET HOST, 5–11
output formatting, 5–11, 5–27
output processing, 5–10
page length and width, 5–41, 5–79
parity flag, 5–42
PASTHRU mode, 5–9, 5–11, 5–26, 5–28
process preservation, 5–17
programming examples, 5–85
protocol, 5–14
read function, 5–28 to 5–37
read verify, 5–7, 5–36
receive speed, 5–42
redisplay data (Ctrl/R), 5–6, 5–28
ReGIS graphics, 5–26
restart data (Ctrl/Q), 5–7
sense characteristics function, 5–78
sense mode function, 5–78
serial line multiplexer, 5–1
set characteristics function, 5–40
arguments, 5–40
set mode function, 5–40
arguments, 5–40
SET TERMINAL DCL command, 5–4, 5–19, 5–26
SIXEL graphics, 5–26
special operating modes, 5–10
status (Ctrl/T), 5–7
status returns, A–7
stop data (Ctrl/S), 5–7
supported devices, 5–1
SYS$GETDVI returns, 5–20
system password, 5–26
tab
Ctrl/I, 5–7
mechanical, 5–22
stops, 5–37
terminator mask, 5–30, 5–31
time (Ctrl/T), 5–7
transmit speed, 5–42
TTY_DIALTYPE SYSGEN parameter, 5–13, 5–16, 5–17
type-ahead, 5–9, 5–18, 5–21, 5–79
alternate buffer, 5–23
unsolicited data, 5–18

Terminals (cont'd)
write breakthrough function, 5–38
write function, 5–37 to 5–40
XON/XOFF control, 5–26
Terminator character bit mask, 5–30
Thrashing magnetic tape, 3–7
Token Ring
Address mappings, 8–8
Translations
logical to physical, 2–15
Truncate subfunction, 1–12
TU58 magnetic tapes
See Disks

U

UDA50 disk adapter, 2–1
Unload function
disk, 2–36
magnetic tape, 3–18

W

Write attention AST function, 4–12
Write breakthrough function, 5–38
Write end-of-file function
magnetic tape, 3–17
message, 4–11
Write protection
hardware, 7–2

X

XQP (extended QIO processor), 1–1