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

Order Number: AA–Z600C–TE

April 1986

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

Revision/Update Information:	This document supersedes the *VAX/VMS I/O User's Reference Manual: Part I* Version 4.0 and Version 4.2.
Software Version:	VAX/VMS Version 4.4

digital equipment corporation
maynard, massachusetts

April 1986

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

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

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

Copyright ©1986 by Digital Equipment Corporation

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

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

The following are trademarks of Digital Equipment Corporation:

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

digital

ZK-3037

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*	**CANADA**	**INTERNATIONAL**
Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario K2K 2A6 Attn: Direct Order Desk	Digital Equipment Corporation PSG Business Manager c/o Digital's local subsidiary or approved distributor

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

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

In Canada call 800-267-6215.

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

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

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TeX, the typesetting system developed by Donald E. Knuth at Stanford University. TeX is a registered trademark of the American Mathematical Society.

Contents

PREFACE xv

SUMMARY OF TECHNICAL CHANGES xix

SECTION 1 ACP–QIO INTERFACE 1-1

1.1 ACP FUNCTIONS AND ENCODING 1-2

1.2 FILE INFORMATION BLOCK (FIB) 1-3

1.3 ACP SUBFUNCTIONS 1-8
- 1.3.1 Directory Lookup 1-9
  - 1.3.1.1 Input Parameters • 1-9
  - 1.3.1.2 Operation • 1-10
- 1.3.2 Access 1-11
  - 1.3.2.1 Input Parameters • 1-11
  - 1.3.2.2 Operation • 1-12
- 1.3.3 Extend 1-12
  - 1.3.3.1 Input Parameters • 1-13
  - 1.3.3.2 Operation • 1-15
- 1.3.4 Truncate 1-15
  - 1.3.4.1 Input Parameters • 1-15
  - 1.3.4.2 Operation • 1-16
- 1.3.5 Read/Write Attributes 1-16
  - 1.3.5.1 Input Parameters • 1-16

1.4 ACP QIO RECORD ATTRIBUTES AREA 1-21

1.5 ACP–QIO ATTRIBUTES STATISTICS BLOCK 1-23

1.6 MAJOR FUNCTIONS 1-24
- 1.6.1 Create File 1-25
  - 1.6.1.1 Input Parameters • 1-25
  - 1.6.1.2 Disk ACP Operation • 1-27
  - 1.6.1.3 Magnetic Tape ACP Operation • 1-28
- 1.6.2 Access File 1-28
  - 1.6.2.1 Input Parameters • 1-29
  - 1.6.2.2 Operation • 1-30

1.6.3 **Deaccess File** 1-31
1.6.3.1 Input Parameters • 1-31
1.6.3.2 Operation • 1-31
1.6.4 **Modify File** 1-31
1.6.4.1 Input Parameters • 1-32
1.6.4.2 Operation • 1-32
1.6.5 **Delete File** 1-33
1.6.5.1 Operation • 1-33
1.6.6 **Mount** 1-33
1.6.7 **ACP Control** 1-34
1.6.7.1 Input Parameters • 1-34
1.6.7.2 Magnetic Tape Control Functions • 1-35
1.6.7.3 Miscellaneous Disk Control Functions • 1-36
1.6.7.4 Disk Quotas • 1-36

1.7 I/O STATUS BLOCK 1-39

SECTION 2 CARD READER DRIVER 2-1

2.1 SUPPORTED CARD READER DEVICE 2-1

2.2 DRIVER FEATURES AND CAPABILITIES 2-1
2.2.1 **Read Modes** 2-1
2.2.2 **Special Card Punch Combinations** 2-1
2.2.2.1 End-of-File Condition • 2-2
2.2.2.2 Set Translation Mode • 2-2
2.2.3 **Error Recovery** 2-2

2.3 DEVICE INFORMATION 2-3

2.4 CARD READER FUNCTION CODES 2-4
2.4.1 **Read** 2-4
2.4.2 **Sense Card Reader Mode** 2-6
2.4.3 **Set Mode** 2-6
2.4.3.1 Set Mode • 2-6
2.4.3.2 Set Characteristic • 2-9

2.5 I/O STATUS BLOCK 2-9

SECTION 3 DISK DRIVERS 3-1

3.1 SUPPORTED DISK DEVICES AND CONTROLLERS 3-1
3.1.1 UDA50 UNIBUS Disk Adapter 3-2
3.1.2 KDA50 Disk Controller 3-2
3.1.3 KDB50 Disk Controller 3-3
3.1.4 HSC50 Controller 3-3
3.1.5 RA60 Pack Disk 3-3
3.1.6 RB02 and RL02 Cartridge Disks 3-4
3.1.7 RM03 and RM05 Pack Disks 3-4
3.1.8 RA80/R80/RM80 and RA81 Fixed Media Disks 3-4
3.1.9 RP05 and RP06 Pack Disks 3-4
3.1.10 RP07 Fixed Media Disk 3-5
3.1.11 RK06 and RK07 Cartridge Disks 3-5
3.1.12 RC25 Disk 3-5
3.1.13 RRD50 Read-Only Memory (CDROM) 3-5
3.1.14 RX01 Console Disk 3-5
3.1.15 RX02 Floppy Disk 3-6
3.1.16 TU58 Magnetic Tape (DECtape II) 3-6

3.2 DRIVER FEATURES AND CAPABILITIES 3-6
3.2.1 Dual Porting (MASSBUS) 3-7
3.2.1.1 Port Selection and Access Modes • 3-7
3.2.1.2 Disk Use and Restrictions • 3-8
3.2.1.3 Dual-Porting DSA Disks • 3-8
3.2.2 Data Check 3-9
3.2.3 Overlapped Seeks 3-9
3.2.4 Error Recovery 3-10
3.2.4.1 Skip Sectoring • 3-10
3.2.5 Logical-to-Physical Translation (RX01 and RX02) 3-11
3.2.6 DIGITAL Storage Architecture (DSA) Devices 3-12
3.2.6.1 Bad Block Replacement and Forced Errors for DSA Disks • 3-13

3.3 DEVICE INFORMATION 3-14

3.4 DISK FUNCTION CODES 3-15
3.4.1 Read 3-20
3.4.2 Write 3-21
3.4.3 Sense Mode 3-22
3.4.4 Set Density 3-22
3.4.5 Search 3-23
3.4.6 Pack Acknowledge 3-23
3.4.7 Unload 3-24

3.4.8 Available 3-24
3.4.9 Seek 3-24
3.4.10 Write Check 3-24

3.5 I/O STATUS BLOCK 3-25

3.6 PROGRAMMING EXAMPLE 3-25

SECTION 4 LABORATORY PERIPHERAL ACCELERATOR DRIVER 4-1

4.1 SUPPORTED DEVICE 4-1
4.1.1 LPA11-K Modes of Operation 4-1
4.1.2 Errors 4-2

4.2 SUPPORTING SOFTWARE 4-2

4.3 DEVICE INFORMATION 4-3

4.4 LPA11-K FUNCTION CODES 4-6
4.4.1 Load Microcode 4-7
4.4.2 Start Microprocessor 4-7
4.4.3 Initialize LPA11-K 4-8
4.4.4 Set Clock 4-8
4.4.5 Start Data Transfer Request 4-9
4.4.6 LPA11-K Data Transfer Stop Command 4-12

4.5 HIGH-LEVEL LANGUAGE INTERFACE 4-12
4.5.1 High-Level Language Support Routines 4-13
4.5.1.1 Buffer Queue Control • 4-13
4.5.1.2 Subroutine Argument Usage • 4-14
4.5.2 LPA$ADSWP — Initiate Synchronous A/D Sampling Sweep 4-18
4.5.3 LPA$DASWP — Initiate Synchronous D/A Sweep 4-19
4.5.4 LPA$DISWP — Initiate Synchronous Digital Input Sweep 4-19
4.5.5 LPA$DOSWP — Initiate Synchronous Digital Output Sweep 4-20
4.5.6 LPA$LAMSKS — Set LPA11-K Masks and NUM Buffer 4-21
4.5.7 LPA$SETADC — Set Channel Information for Sweeps 4-22
4.5.8 LPA$SETIBF — Set IBUF Array for Sweeps 4-22

4.5.9 LPA$STPSWP — Stop In-Progress Sweep 4-23
4.5.10 LPA$CLOCKA — Clock A Control 4-24
4.5.11 LPA$CLOCKB — Clock B Control 4-25
4.5.12 LPA$XRATE — Compute Clock Rate and Preset Value 4-25
4.5.13 LPA$IBFSTS — Return Buffer Status 4-26
4.5.14 LPA$IGTBUF — Return Buffer Number 4-27
4.5.15 LPA$INXTBF — Set Next Buffer To Use 4-28
4.5.16 LPA$IWTBUF — Return Next Buffer or Wait 4-28
4.5.17 LPA$RLSBUF — Release Data Buffer 4-29
4.5.18 LPA$RMVBUF — Remove Buffer from Device Queue 4-30
4.5.19 LPA$CVADF — Convert A/D Input to Floating-Point 4-30
4.5.20 LPA$FLT16 — Convert Unsigned 16-bit Integer to Floating-Point 4-30
4.5.21 LPA$LOADMC — Load Microcode and Initialize LPA11-K 4-31

4.6 I/O STATUS BLOCK 4-31

4.7 LOADING LPA11-K MICROCODE 4-32
4.7.1 Microcode Loader Process 4-32
4.7.2 Operator Process 4-33

4.8 RSX–11M/M–PLUS AND VAX/VMS DIFFERENCES 4-33
4.8.1 Alignment and Length 4-34
4.8.2 Status Returns 4-34
4.8.3 Sweep Routines 4-34
4.8.4 General 4-34

4.9 PROGRAMMING EXAMPLES 4-35
4.9.1 LPA11-K High-Level Language Program (Program A) 4-35
4.9.2 LPA11-K High-Level Language Program (Program B) 4-37
4.9.3 LPA11-K QIO Functions Program (Program C) 4-41

SECTION 5 LINE PRINTER DRIVER 5-1

5.1 SUPPORTED LINE PRINTER DEVICES 5-1
5.1.1 LP11 Line Printer Controller 5-1
5.1.2 DMF32 and DMB32 Line Printer Controllers 5-1
5.1.3 LP27 Line Printer 5-1
5.1.4 LA11 DECprinter I 5-2
5.1.5 LN01 Laser Page Printer 5-2
5.1.6 LN03 Laser Page Printer 5-2

5.2 DRIVER FEATURES AND CAPABILITIES 5-2
5.2.1 Output Character Formatting 5-2
5.2.2 Error Recovery 5-3

5.3 DEVICE INFORMATION 5-3

5.4 LINE PRINTER FUNCTION CODES 5-5
5.4.1 Write 5-5
5.4.1.1 Write Function Carriage Control • 5-6
5.4.2 Sense Printer Mode 5-8
5.4.3 Set Mode 5-9

5.5 I/O STATUS BLOCK 5-10

5.6 PROGRAMMING EXAMPLE 5-10

SECTION 6 MAGNETIC TAPE DRIVERS 6-1

6.1 SUPPORTED MAGNETIC TAPE CONTROLLERS 6-2
6.1.1 TM03 Magnetic Tape Controller 6-2
6.1.2 TS11 Magnetic Tape Controller 6-2
6.1.3 TM78 Magnetic Tape Controller 6-2
6.1.4 TU80 Magnetic Tape Subsystem 6-2
6.1.5 TU81 and TA81 Magnetic Tape Subsystems 6-2
6.1.6 TK50 Cartridge Tape System 6-3

6.2 DRIVER FEATURES AND CAPABILITIES 6-3
6.2.1 Master Adapters and Slave Formatters 6-3
6.2.2 Data Check 6-4
6.2.3 Error Recovery 6-4
6.2.4 Streaming Tape Systems 6-5

6.3 DEVICE INFORMATION 6-6

6.4 MAGNETIC TAPE FUNCTION CODES 6-8
6.4.1 Read 6-12
6.4.2 Write 6-13
6.4.3 Rewind 6-14
6.4.4 Skip File 6-14
6.4.5 Skip Record 6-15
6.4.5.1 Logical End-of-Volume Detection • 6-16

6.4.6 Write End-of-File 6-16
6.4.7 Rewind Offline 6-16
6.4.8 Unload 6-17
6.4.9 Sense Tape Mode 6-17
6.4.10 Set Mode 6-18
6.4.11 Data Security Erase 6-21
6.4.12 Pack Acknowledge 6-21
6.4.13 Available 6-21

6.5 I/O STATUS BLOCK 6-22

6.6 PROGRAMMING EXAMPLES 6-22

SECTION 7 MAILBOX DRIVER 7-1

7.1 MAILBOX OPERATIONS 7-1
7.1.1 Creating Mailboxes 7-2
7.1.2 Deleting Mailboxes 7-3
7.1.3 Mailbox Message Format 7-4
7.1.4 Mailbox Protection 7-4

7.2 DEVICE INFORMATION 7-5

7.3 MAILBOX FUNCTION CODES 7-6
7.3.1 Read 7-6
7.3.2 Write 7-7
7.3.3 Write End-of-File Message 7-8
7.3.4 Set Attention AST 7-8
7.3.5 Set Protection 7-10

7.4 I/O STATUS BLOCK 7-11

7.5 PROGRAMMING EXAMPLE 7-12

SECTION 8 TERMINAL DRIVER 8-1

8.1 SUPPORTED TERMINAL DEVICES 8-1

8.2 TERMINAL DRIVER FEATURES AND CAPABILITIES 8-2
- **8.2.1 Input Processing** 8-2
  - 8.2.1.1 Command Line Editing and Command Recall • 8-3
  - 8.2.1.2 Control Characters and Special Keys • 8-3
  - 8.2.1.3 Read Verify • 8-7
  - 8.2.1.4 Escape and Control Sequences • 8-7
  - 8.2.1.5 Type-Ahead Capability • 8-8
  - 8.2.1.6 Line Terminators • 8-9
  - 8.2.1.7 Special Operating Modes • 8-10
- **8.2.2 Output Processing** 8-10
  - 8.2.2.1 Duplex Modes • 8-10
  - 8.2.2.2 Formatting of Output • 8-11
- **8.2.3 Dial-Up Support** 8-11
  - 8.2.3.1 Modem Signal Control • 8-11
  - 8.2.3.2 Hangup on Logging Out • 8-14
  - 8.2.3.3 Preservation of a Process Across Hangups • 8-14
- **8.2.4 Terminal/Mailbox Interaction** 8-16
- **8.2.5 Autobaud Detection** 8-17
- **8.2.6 Out-of-Band Control Character Handling** 8-18

8.3 DEVICE INFORMATION 8-18
- **8.3.1 Terminal Characteristics Categories** 8-24

8.4 TERMINAL FUNCTION CODES 8-25
- **8.4.1 Read** 8-25
  - 8.4.1.1 Function Modifier Codes for Read QIO Functions • 8-27
  - 8.4.1.2 Read Function Terminators • 8-28
  - 8.4.1.3 Itemlist Read Operations • 8-28
  - 8.4.1.4 Read Verify Function • 8-33
- **8.4.2 Write** 8-34
  - 8.4.2.1 Function Modifier Codes for Write QIO Functions • 8-35
  - 8.4.2.2 Write Function Carriage Control • 8-35
- **8.4.3 Set Mode** 8-37
  - 8.4.3.1 Hangup Function Modifier • 8-40
  - 8.4.3.2 Enable CTRL/C AST and Enable CTRL/Y AST Function Modifiers • 8-41
  - 8.4.3.3 Set Modem Function Modifier • 8-42
  - 8.4.3.4 Loopback Function Modifier • 8-42
  - 8.4.3.5 Enable Out-of-Band AST Function Modifier • 8-43
  - 8.4.3.6 Broadcast Function Modifier • 8-45
- **8.4.4 Sense Mode and Sense Characteristics** 8-45
  - 8.4.4.1 Type-ahead Count Function Modifier • 8-46
  - 8.4.4.2 Read Modem Function Modifier • 8-47
  - 8.4.4.3 Broadcast Function Modifier • 8-48

8.5 I/O STATUS BLOCK 8-48

8.6 PROGRAMMING EXAMPLES 8-50

APPENDIX A I/O FUNCTION CODES A-1

A.1 ACP-QIO INTERFACE DRIVER A-1

A.2 CARD READER DRIVER A-2

A.3 DISK DRIVERS A-2

A.4 LABORATORY PERIPHERAL ACCELERATOR DRIVER A-3

A.5 LINE PRINTER DRIVER A-5

A.6 MAGNETIC TAPE DRIVERS A-5

A.7 MAILBOX DRIVER A-7

A.8 TERMINAL DRIVER A-7

APPENDIX B TABLES B-1

B.1 DEC MULTINATIONAL CHARACTER SET B-1

B.2 TERMINAL SEQUENCES AND MODES B-9

INDEX

EXAMPLES

3–1 Disk Program Example 3-26
4–1 LPA11-K High-Level Language Program (Program A) 4-36
4–2 LPA11-K High-Level Language Program (Program B) 4-38

4–3 LPA11-K QIO Functions Program (Program C) 4-42
5–1 Line Printer Program Example 5-11
6–1 Magnetic Tape Program Example 6-23
6–2 Device Characteristic Program Example 6-26
6–3 Set Mode and Sense Mode Program Example 6-27
7–1 Mailbox Driver Program Example 7-13
8–1 Terminal Program Example 8-51
8–2 Read Verify Program Example 8-58

FIGURES

1–1 ACP–QIO Interface 1-1
1–2 ACP Device/Function-Dependent Arguments 1-3
1–3 ACP Device/Function Argument Descriptor Format 1-3
1–4 File Information Block Format 1-4
1–5 Typical Short File Information Block 1-5
1–6 Attribute Control Block Format 1-17
1–7 ACP–QIO Record Attributes Area 1-21
1–8 ACP–QIO Attributes Statistics Block 1-23
1–9 Quota File Transfer Block 1-39
1–10 IOSB Contents - ACP–QIO Functions 1-40
2–1 Binary and Packed Column Storage 2-5
2–2 Set Mode Characteristics Buffer 2-6
2–3 Set Characteristic Buffer 2-9
2–4 IOSB Contents 2-9
3–1 Dual-Ported Disk Drives 3-7
3–2 Starting Physical Address 3-20
3–3 Physical Cylinder Number Format 3-20
3–4 IOSB Contents 3-25
3–5 IOSB Contents - Sense Mode 3-25
4–1 Relationship of Supporting Software to LPA11-K 4-4
4–2 Data Transfer Command Table 4-11
4–3 Buffer Queue Control 4-15
4–4 I/O Functions IOSB Content 4-31
5–1 P4 Carriage Control Specifier 5-6
5–2 Write Function Carriage Control (Prefix and Postfix Coding) 5-8
5–3 Set Mode Buffer 5-9
5–4 Set Characteristics Buffer 5-9
5–5 IOSB Contents — Write Function 5-10
5–6 IOSB Contents — Set Mode Function 5-10
6–1 IO$_SKIPFILE Argument 6-15
6–2 IO$_SKIPRECORD Argument 6-15
6–3 Sense Mode P1 Buffer 6-18

6–4 Set Mode Characteristics Buffer 6-19
6–5 Set Characteristics Buffer 6-19
6–6 IOSB Contents 6-22
7–1 Multiple Mailbox Channels 7-3
7–2 Typical Mailbox Message Format 7-4
7–3 Read Mailbox 7-7
7–4 Write Mailbox 7-8
7–5 Write Attention AST (Read Unsolicited Data) 7-9
7–6 Read Attention AST 7-10
7–7 IOSB Contents - Read Function 7-12
7–8 IOSB Contents - Write Function 7-12
7–9 IOSB Contents - Set Protection Function 7-12
8–1 Modem Control - Two-Way Simultaneous Operation 8-15
8–2 Terminal Mailbox Message Format 8-17
8–3 Short and Long Forms of Terminator Mask Quadwords 8-29
8–4 Itemlist Read Descriptor 8-29
8–5 P4 Carriage Control Specifier 8-35
8–6 Write Function Carriage Control (Prefix and Postfix Coding) 8-38
8–7 Set Mode and Set Characteristics Buffers 8-39
8–8 Set Mode P1 Block 8-42
8–9 Relationship of Out-of-Band Function with Control Characters 8-44
8–10 Sense Mode Characteristics Buffer 8-46
8–11 Sense Mode Characteristics Buffer (type-ahead) 8-46
8–12 Sense Mode P1 Block 8-47
8–13 IOSB Contents — Read Function 8-48
8–14 IOSB Contents — Itemlist Read Function 8-49
8–15 IOSB Contents — Write Function 8-49
8–16 IOSB Contents — Set Mode, Set Characteristics, Sense Mode, and Sense Characteristics Functions 8-49

TABLES

1–1 Contents of the File Information Block 1-5
1–2 FIB Fields (Lookup Control) 1-9
1–3 FIB Fields (Access Control) 1-11
1–4 FIB Fields (Extend Control) 1-13
1–5 FIB Fields (Truncate Control) 1-15
1–6 Attribute Control Block Fields 1-17
1–7 ACP–QIO Attributes 1-18
1–8 ACP Record Attributes Values 1-22
1–9 Contents of the Statistics Block 1-23
1–10 Disk Quota Functions (Enable/Disable) 1-37
1–11 Disk Quota Functions (Individual Entries) 1-38
2–1 Card Reader Device-Independent Characteristics 2-3
2–2 Device-Dependent Characteristics for Card Readers 2-4

2–3 Card Reader I/O Functions 2-4
2–4 Set Mode and Set Characteristic Card Reader Characteristics 2-6
2–5 Card Reader Codes 2-7
3–1 Supported Disk Devices 3-1
3–2 Disk Device Characteristics 3-14
3–3 Disk I/O Functions 3-16
4–1 Minimum and Maximum Configurations per LPA11-K 4-1
4–2 LPA11-K Device-Independent Characteristics 4-4
4–3 LPA11-K Device-Dependent Characteristics 4-5
4–4 VAX Procedures for the LPA11-K 4-13
4–5 Subroutine Argument Usage 4-15
4–6 LPA$IGTBUF Call — IBUFNO and IOSB Contents 4-27
4–7 LPA$IWTBUF Call — IBUFNO and IOSB Contents 4-29
4–8 Program A Variables 4-35
4–9 Program B Variables 4-37
5–1 Printer Device-Independent Characteristics 5-4
5–2 Device-Dependent Characteristics for Line Printers 5-4
5–3 Write Function Carriage Control (FORTRAN: byte 0 not equal to 0) 5-6
5–4 Write Function Carriage Control (P4 byte 0 equal to 0) 5-7
6–1 Supported Magnetic Tape Devices 6-1
6–2 Magnetic Tape Device-Independent Characteristics 6-6
6–3 Device-Dependent Information for Tape Devices 6-6
6–4 Extended Device Characteristics for Tape Devices 6-7
6–5 Magnetic Tape I/O Functions 6-8
6–6 Set Mode and Set Characteristics Magnetic Tape Characteristics 6-20
6–7 Extended Device Characteristics for Tape Devices 6-20
7–1 Mailbox Read and Write Operations 7-1
7–2 Mailbox Characteristics 7-5
8–1 Supported Terminal Devices 8-1
8–2 Terminal Control Characters 8-4
8–3 Control and Data Signals (Full Modem Mode Configuration) 8-12
8–4 Terminal Device-Independent Characteristics 8-19
8–5 Terminal Characteristics 8-19
8–6 Extended Terminal Characteristics 8-21
8–7 Read QIO Function Modifiers for the Terminal Driver 8-27
8–8 Item Codes for Itemlist Read Operations for the Terminal Driver 8-30
8–9 Write QIO Function Modifiers for the Terminal Driver 8-35
8–10 Write Function Carriage Control (FORTRAN: byte 0 not equal to 0) 8-36
8–11 Write Function Carriage Control (P4 byte 0 = 0) 8-37
8–12 Broadcast Requester IDs 8-45
B–1 DEC Multinational Character Set B-1
B–2 Sequences and Modes B-9

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 the I/O devices. Users of VAX/VMS who do not require such detailed knowledge of I/O drivers can use the device-independent services described in the *VAX Record Management Services Reference Manual.* Readers are expected to have some experience with either VAX MACRO or some other high-level assembly language.

Structure of This Document

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

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

Associated Documents

The following documents may also be useful.

- *General Information Volume* — contains a complete list of all VAX/VMS documents and a master index of all topics discussed in the document set
- *VAX/VMS System Services Reference Manual*
- *VAX Software Handbook*
- *PDP-11 Peripherals Handbook*
- *VAX FORTRAN User's Guide*

- *Guide to Programming on VAX/VMS*
- *VAX Record Management Services Reference Manual*
- *LPA11-K Laboratory Peripheral Accelerator User's Guide*
- *VAX/VMS Networking Manual*
- *VAX/VMS System Messages and Recovery Procedures Reference Manual*

Conventions Used in This Document

Convention	Meaning
<>	Angle brackets enclose keys that appear on the terminal keyboard. For example: `<0> <20-2F>...<40-7E>`
RET	A symbol with a one- to three-character abbreviation indicates that you press a key on the terminal, for example, RET .
CTRL/x	The phrase CTRL/x indicates that you must press the key labeled CTRL while you simultaneously press another key, for example, CTRL/C, CTRL/Y, CTRL/O.
. . .	Vertical series of periods, or ellipsis, mean either that not all the data that the system would display in response to the particular command is shown or that not all the data a user would enter is shown. Vertical ellipsis in coding examples indicate that lines of code not pertinent to the example are omitted. For example:

```
TTNAME:    .ASCID /_TTB4/
TTCHAN:    .BLKW 1
              .
              .
              .
     $ASSIGN_S DEVNAM=TTNAME,CHAN=TTCHAN
```

Convention	Meaning
[ ]	Brackets in QIO requests enclose optional arguments. For example: `IO$_SETCHAR P1,[P2],P3,[P6]`
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

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

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

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

Summary of Technical Changes

This revision of the *VAX/VMS I/O User's Reference Manual: Part I* contains the technical changes since VAX/VMS Version 4.2. The following sections contain new or changed information:

- Section 2—This section on the card reader driver now includes information on the Get Device/Volume Information ($GETDVI) system service, which replaces the Get I/O Channel Information ($GETCHN) and $Get I/O Device Information (GETDEV) system services.
- Section 3—This section on the disk drivers now includes information on the KDA50 disk controller and the RRD50 read-only memory (CDROM).
- Section 4—This section on the LPA11-K driver now includes information on the $GETDVI system service, which replaces the $GETCHN and $GETDEV system services.
- Section 5—This section on the line printer driver includes information on the $GETDVI system service, which replaces the $GETCHN and $GETDEV system services, the DMB32 line printer controller, and the LN03 laser page printer.
- Section 6—This section on the magnetic tape drivers now includes information on the $GETDVI system service, which replaces the $GETCHN and $GETDEV system services, and the TK50 and TU81E magnetic tapes.
- Section 7—This section on the mailbox driver now includes information on the $GETDVI system service, which replaces the $GETCHN and $GETDEV system services.
- Section 8—This section on the terminal driver now includes information on the $GETDVI system service, which replaces the $GETCHN and $GETDEV system services. The following terminal driver features are also described:
  - VAX 8200 processor support
  - New set mode P5 function modifier to change frame size
  - Updated section on automatic baud rate detection
  - New timeout feature for remote terminal connections
  - Updated section on escape and control sequences

1 ACP–QIO Interface

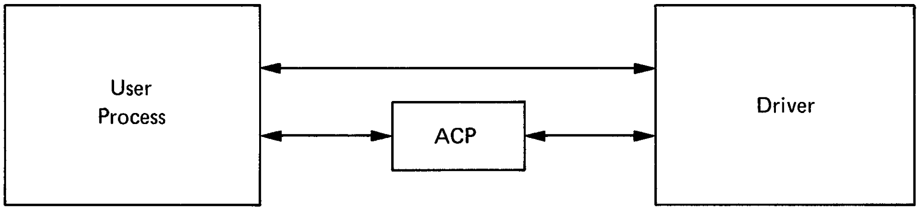
An ancillary control process (ACP) is a process that interfaces between the user process and the driver, and performs functions that supplement the driver's functions. Virtual I/O operations involving file-structured devices (disks and magnetic tapes) often require ACP intervention. In most cases, ACP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

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

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

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

Figure 1–1 ACP–QIO Interface



ZK-635-82

This section also describes a number of structures and field names of the form xxx$name. A VAX MACRO program may define symbols of this form by invoking the $xxxDEF 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 VAX BLISS-32 may use these symbols simply by referencing them and including the correct library, that is, SYS$LIBRARY:STARLET.L32 and SYS$LIBRARY:LIB.L32, respectively, for the symbol classes listed previously.

Users should note that references to ANSI refer to the **American National Standard Magnetic Tape Labels and File Structures for Information Interchange, ANSI X3.27-1978.**

1.1 ACP Functions and Encoding

All VAX/VMS 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/or file allocation
- IO$_DELETE—deletes a directory entry and/or file header
- IO$_MOUNT—informs the ACP when a volume is mounted; requires MOUNT privilege
- IO$_ACPCONTROL—performs miscellaneous control functions

The function modifiers are:

- IO$M_ACCESS—opens the file on the user's channel
- IO$M_CREATE—creates a file
- IO$M_DELETE—deletes the file (or marks it for deletion)
- IO$M_DMOUNT—dismounts a volume

In addition to the function codes and modifiers, VAX/VMS ACPs take five device/function-dependent arguments, as shown in Figure 1–2.

The first argument, P1, is the address of the file information block (FIB) descriptor. Section 1.2 describes the FIB in detail.

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

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

Figure 1–2 ACP Device/Function-Dependent Arguments

	31 0
P1:	Address of FIB descriptor
P2:	Address of file name string descriptor (optional)
P3:	Address of word to receive resultant string length (optional)
P4:	Address of resultant string descriptor (optional)
P5:	Address of attribute control block (optional)

ZK-636-82

Figure 1–3 ACP Device/Function Argument Descriptor Format

31 16	15 0
not used	count
address	

ZK-637-82

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

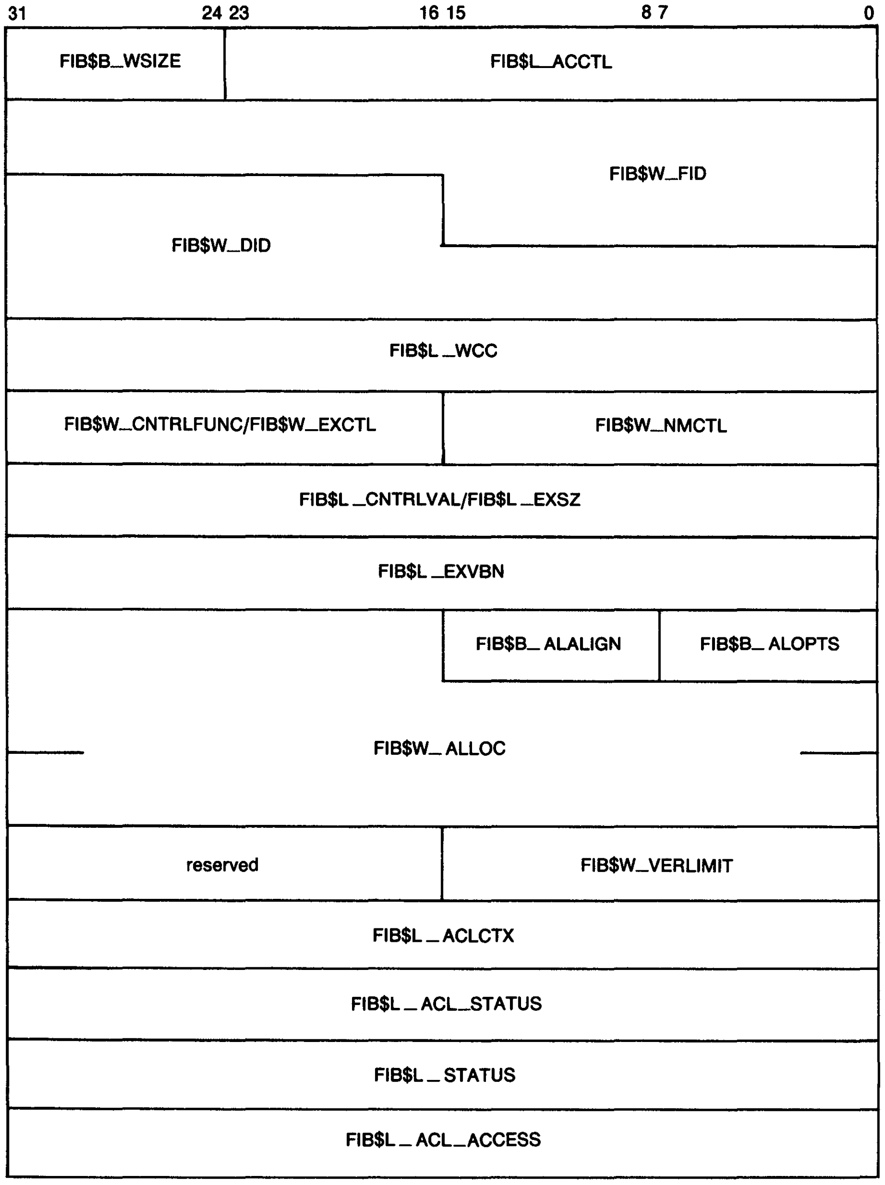
All areas of memory specified by the descriptors must be read/write.

Figure 1–3 shows the format for the descriptors.

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 writeable. Because the FIB is passed by a descriptor, its length can vary. Thus a short FIB can be used in ACP calls that do not need arguments toward the end of the FIB. The ACP treats the omitted portion of the FIB as if it were 0. Figure 1–5 shows the format of a typical short FIB, in this case one 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, where the individual functions are described.

Figure 1–4 File Information Block Format


31
24 23
16 15
8 7
0
FIB$B_WSIZE
FIB$L_ACCTL
FIB$W_FID
FIB$W_DID
FIB$L_WCC
FIB$W_CNTRLFUNC/FIB$W_EXCTL
FIB$W_NMCTL
FIB$L_CNTRLVAL/FIB$L_EXSZ
FIB$L_EXVBN
FIB$B_ALALIGN
FIB$B_ALOPTS
FIB$W_ALLOC
reserved
FIB$W_VERLIMIT
FIB$L_ACLCTX
FIB$L_ACL_STATUS
FIB$L_STATUS
FIB$L_ACL_ACCESS

ZK-638-82

Figure 1–5 Typical Short File Information Block

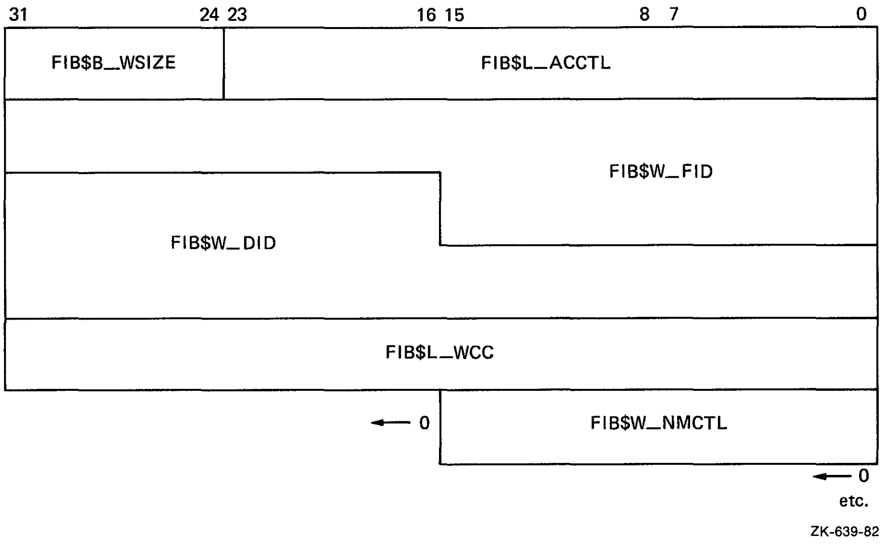


Table 1–1 Contents of the File Information Block

Field	Subfields	Meaning
FIB$L_ACCTL		Contains flag bits that control the access to the file. Sections 1.3.1.1, 1.3.2.1, 1.6.1.1, 1.6.4.1, and 1.6.5 describe the FIB$L_ACCTL field flag bits.
FIB$B_WSIZE		Controls the size of the file window used to map a disk file. If a window size of 255 is specified, the file is mapped completely through the use of segmented windows.
FIB$W_FID		Specifies the file identification. The user supplies the file identifier when it is known; the ACP returns the file identifier when it becomes known, for example, as a result of a create or directory lookup. A 0 file identifier may be specified when an operation is performed on a file that is already open on a particular channel. The ACP will return the file identifier of the open file. The following subfields are defined:

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

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

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

Field	Subfields	Meaning
FIB$L_EXSZ		Specifies the number of blocks to be allocated in an extend operation on a disk file.
FIB$L_CNTRLVAL		Contains a control function value used in an IO$_ACPCONTROL function (see Section 1.6.7). The interpretation of the value depends on the control function specified in FIB$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 RVN or placement RVN.
	FIB$B_LOC_NMX	Related file number extension.
	FIB$L_LOC_ADDR	Placement LBN, cylinder, or 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.

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

Field	Subfields	Meaning
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 and checks whether it is deleteable. The mask has the same configuration as the standard protection mask.

1.3 ACP Subfunctions

The operations that the ACP performs may 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, for example, IO$_ACCESS, IO$_CREATE, or IO$_MODIFY. While executing the major function, one or more subfunctions may be performed. A subfunction is an operation such as looking up, accessing, or extending a file. Most subfunctions may be executed by more than one of the major functions. Sections 1.3.1 through 1.3.5 describe the following subfunctions in detail:

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

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

1.3.1 Directory Lookup

The directory lookup subfunction is used to search for a file in a disk directory or on a magnetic tape. This subfunction may be invoked by the major functions IO$_ACCESS, IO$_MODIFY, IO$_DELETE, and certain functions of IO$_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	Field Values	Meaning
FIB$W_NMCTL		Name string control. The following name control bits are applicable to a lookup operation:
	FIB$M_WILD	Set if name string contains wildcards. Setting this bit causes wildcard context to be returned in FIB$L_WCC.
	FIB$M_ALLNAM	Set to match all name field values.
	FIB$M_ALLTYP	Set to match all field type values.
	FIB$M_ALLVER	Set to match all version field values.
	FIB$M_FINDFID	Set to search a directory for the file identifier in FIB$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.
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$M_REWIND	Set to rewind magnetic tape before lookup. If not set, a magnetic tape will be searched from its current position.

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

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

```
name.type;version
name.type.version
```

The name and type may 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 may be present.

If any of the bits FIBM_ALLNAM, FIBM_ALLTYP, and FIB$M_ALLVER are set, then the contents of the corresponding field in the name string are ignored and assumed to be a single 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 VAX 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. Both the P3 and P4 arguments are optional.

1.3.1.2 Operation

The system searches 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 may be used in subsequent operations as the major function is processed.

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

Wildcard lookups are performed by specifying the appropriate wildcard characters in the name string and setting FIB$M_WILD. (The name control bits FIBM_ALLNAM, FIBM_ALLTYP, and FIB$M_ALLVER may also be used in searching for wildcard entries, but they are intended primarily for compatibility mode use.) On the first lookup, FIB$L_WCC should contain zero entries. On each lookup, the ACP will return a nonzero value in FIB$L_WCC, which must be passed back on the next lookup call. In addition, the user 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.

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

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

1.3.2 Access

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

1.3.2.1 Input Parameters

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

Table 1–3 FIB Fields (Access Control)

Field	Field Values	Meaning
FIB$L_ACCTL		Specifies field values that control access to the file. The following access control bits are applicable to the access subfunction:
	FIB$M_WRITE	Set for write access; clear for read-only access.
	FIB$M_NOREAD	Set to deny read access to others. (The user must have write privilege to the file to use this option.)
	FIB$M_NOWRITE	Set to deny write access to others.
	FIB$M_NOTRUNC	Set to prevent the file from being truncated; clear to allow truncation.
	FIB$M_DLOCK	Set to enable deaccess lock (close check). Used only for disk devices. Used to flag a file as inconsistent if the program currently modifying the file terminates abnormally. If the program then deaccesses the file without performing a write attributes operation, the file is marked as locked and cannot be accessed until it is unlocked.
	FIB$M_UPDATE	Set to position at start of a magnetic tape file when opening file for write; clear to position at end-of-file.
	FIB$M_READCK	Set to enable read checking of the file. Virtual reads to the file will be performed using a data check operation.
	FIB$M_WRITECK	Set to enable write checking of the file. Virtual writes to the file will be performed using a data check operation.

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

Field	Field Values	Meaning
	FIB$M_EXECUTE	Set to access the file in execute mode. The protection check is made against the EXECUTE bit instead of the READ bit. Valid only for requests issued from SUPERVISOR, EXEC, or KERNEL mode.
	FIB$M_NOLOCK	Set to override exclusive access to the file, that is, to access to the file even though another user has the file open with FIB$M_NOREAD specified. The user must have SYSPRV privilege or ownership of the volume to use this option. FIB$M_NOREAD and FIB$M_NOWRITE must be clear for this option to work.
	FIB$M_NORECORD	Set to inhibit recording of the file's expiration date. If not set, the file's expiration date may be modified, depending on the file retention parameters of the volume.
FIB$B_WSIZE		Controls the size of the file window used to map a disk file. The ACP will use the volume default if FIB$B_WSIZE is 0. A value of 1 to 127 indicates the number of retrieval pointers to be allocated to the window. A value of –1 indicates that the window should be as large as necessary to map the entire file. 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 may be invoked by 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$M_EXTEND is set in the extend control word FIB$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	Field Values	Meaning
FIB$W_EXCTL		Extend control flags. The following flags are applicable to the extend subfunction:
	FIB$M_EXTEND	Set to enable extension.
	FIB$M_NOHDREXT	Set to inhibit generation of extension file headers.
	FIB$M_ALCON	Allocates contiguous space. The extend operation will fail if the needed contiguous space is not available.
	FIB$M_ALCONB	Allocates contiguous space as much as possible. If both FIB$M_ALCON and FIB$M_ALCONB are set, then a single contiguous area, whose size is the largest available but not greater than the size requested, will be allocated.
	FIB$M_FILCON	Marks the file contiguous. This bit may only be set if the file does not already have space allocated to it.
	FIB$M_ALDEF	Allocates the extend size (FIB$L_EXSZ) or the system default, whichever is greater.
FIB$L_EXSZ		Specifies the number of blocks to allocate to the file. The number of blocks actually allocated for this operation is returned in this longword. More blocks than requested may be allocated to meet cluster boundaries.
FIB$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$M_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 will fail.

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

Field	Field Values	Meaning
	FIB$M_ONCYL	Set to locate allocated space within a cylinder. This option only functions correctly when FIB$M_ALCON or FIB$M_ALCONB is specified.
FIB$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$L_LOC_ADDR.
	FIB$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$M_ALCON) fails, the size of the largest contiguous space available on the disk is returned in the second longword of the user's I/O status block.

1.3.4 Truncate

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

1.3.4.1 Input Parameters

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

Table 1–5 FIB Fields (Truncate Control)

Field	Field Values	Meaning
FIB$W_EXCTL		Extend control flags. The following flags are applicable to the truncate subfunction:
	FIB$M_TRUNC	Must be set to enable truncation.
	FIB$M_MARKBAD	Set to append the truncated blocks to the bad block file, instead of returning them to the free storage pool. Only one cluster may be deallocated. This is most easily accomplished by specifying the last VBN of the file in FIB$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$L_EXVBN. Because of cluster round-up, this value may be greater than the value specified. If FIB$M_MARKBAD is specified, the truncation VBN is rounded down instead of up, and the value returned in FIB$L_EXVBN may 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, that is, 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 will be ignored.

1.3.5 Read/Write Attributes

The read and write attributes subfunctions are used, for example, to read and write file protection and creation and revision 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

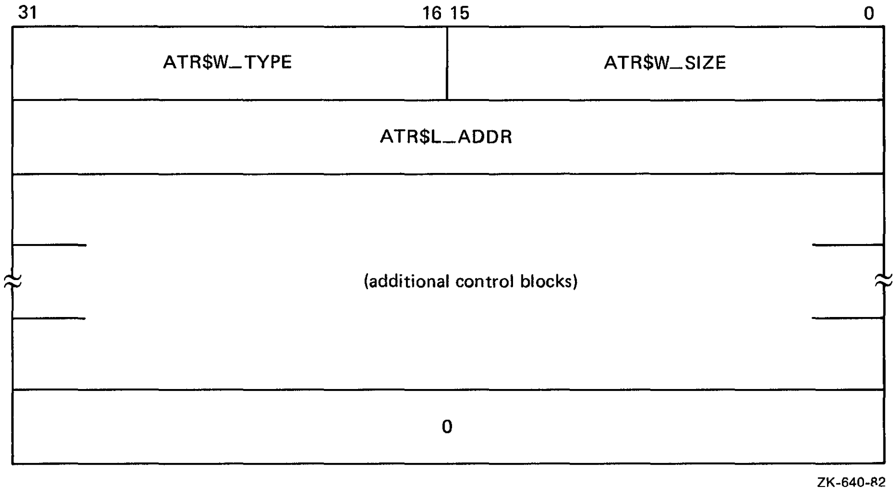


Table 1–6 Attribute Control Block Fields

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 user's memory space to or from which the attribute is to be transferred. The attribute buffer must be writeable.

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

Table 1–7 ACP–QIO Attributes

Attribute Name[1]	Maximum Size (bytes)	Meaning
ATR$C_UCHAR[2,4]	4	4-byte file characteristics. (The file characteristics bits are listed following this table.)
ATR$C_RECATTR[3]	32	Record attribute area. Section 1.4 describes the record attribute area in detail.
ATR$C_FILNAM	10	6-byte Radix–50 file name plus ATR$C_FILTYP and ATR$C_FILVER.
ATR$C_FILTYP	4	2-byte Radix–50 file type plus ATR$C_FILVER.
ATR$C_FILVER	2	2-byte binary version number.
ATR$C_EXPDAT[2]	7	Expiration date in ASCII. Format: DDMMMYY.
ATR$C_STATBLK[5]	32	Statistics block. Section 1.5 describes the statistics block in detail.
ATR$C_HEADER[5]	512	Complete file header.
ATR$C_BLOCKSIZE	2	Magnetic tape block size.
ATR$C_USERLABEL	80	User file label.
ATR$C_ASCDATES[2,4]	35	Revision count (2 binary bytes), revision date, creation date, and expiration date, in ASCII. Format: 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.

---

[1] 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.

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

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

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

[5] Read only.

Table 1–7 (Cont.) ACP–QIO Attributes

Attribute Name[1]	Maximum Size (bytes)	Meaning
ATR$C_ASCNAME	20	Disk: file name, type, and version, in ASCII, including punctuation. Format: name.type;version. Magnetic tape: contains 17-character file identifier (ANSI a); no version number. Overrides all other file name and file type specifications if supplied on input operations. If specified on an access operation and you want only a value to be returned, you must specify (in ATR$W_SIZE) a buffer of greater than 17 bytes.
ATR$C_CREDATE[2]	8	64-bit creation date and time.
ATR$C_REVDATE[2,3]	8	64-bit revision date and time.
ATR$C_EXPDATE[2]	8	64-bit expiration date and time.
ATR$C_BAKDATE[3,9]	8	64-bit backup date and time.
ATR$C_UIC[2]	4	4-byte file owner UIC.
ATR$C_FPRO[2,3]	2	File protection.
ATR$C_RPRO[9]	2	2-byte record protection.
ATR$C_ACLEVEL[2,3,9]	1	File access level.
ATR$C_SEMASK[9]	8	File security mask and limit.
ATR$C_UIC_RO[5]	4	4-byte file owner UIC.
ATR$C_DIRSEQ[9]	2	Directory update sequence count.
ATR$C_BACKLINK[9]	6	File back link pointer.
ATR$C_JOURNAL[9]	2	Journal control flags.
ATR$C_HDR1_ACC	1	ANSI magnetic tape header label accessibility character.
ATR$C_ADDACLENT[6,9,10]	255	Add one or more access control entries.
ATR$C_DELACLENT[6,9,10]	255	Remove an access control entry.
ATR$C_MODACLENT[6,9,10]	255	Modify an ACL entry.
ATR$C_FNDACLENT[9,10]	255	Locate an ACL entry.

---

[1] 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.

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

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

[5] Read only.

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

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

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

Table 1–7 (Cont.) ACP–QIO Attributes

Attribute Name[1]	Maximum Size (bytes)	Meaning
ATR\$C_FNDACETYP[9,10]	255	Find a specific type of ACE.
ATR\$C_DELETEACL[6,9,10]	255	Delete the entire ACL.
ATR\$C_READACL[9,10]	512	Read the entire ACL or as much as will fit in the supplied buffer. Only complete ACEs will be transferred. Thus, the supplied buffer may not be completely filled.
ATR\$C_ACLLENGTH[9,10]	4	Return the length of the ACL.
ATR\$C_READACE[9,10]	255	Read a single ACE.
ATR\$C_RESERVED[8,9]	380	Modify reserve area.
ATR\$C_HIGHWATER[9]	4	High-water mark (user read-only).
ATR\$C_PRIVS_USED[7,9]	4	Privileges used to gain access.
ATR\$C_MATCHING_ACE[7,9]	255	ACE used to gain access (if any).
ATR\$C_ACCESS_MODE	1	Access mode for following attribute descriptors.
ATR\$C_FILE_SPEC[9]	512	Convert FID to file specification.
ATR\$C_BUFFER_OFFSET[4]	2	Offset length for ANSI magnetic tape header label buffer.

[1] 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.

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

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

[7] This attribute may only be retrieved on the initial file access or create operation.

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

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

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

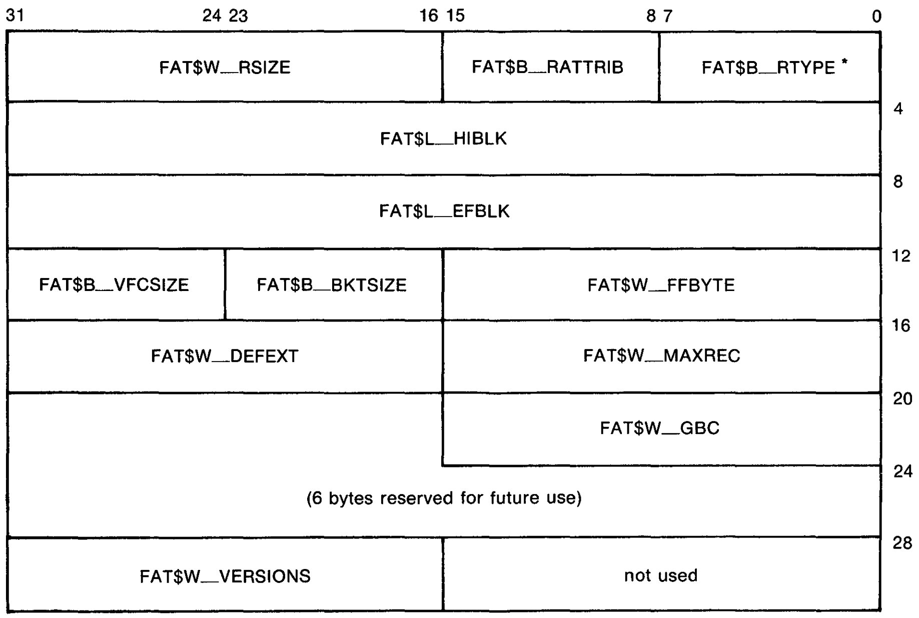
The file characteristics longword (read with the ATR$C_UCHAR attribute) contains the following characteristics bits:

FCHNOBACKUP	File is not to be backed up.
FCH$M_READCHECK	Verify all read operations.
FCH$M_WRITCHECK	Verify all write operations.
FCH$M_CONTIGB	Keep file as contiguous as possible.
FCH$M_LOCKED	File is deaccess-locked.
FCH$M_CONTIG	File is contiguous.
FCH$M_BADACL	File's ACL is corrupt.
FCH$M_SPOOL	File is an intermediate spool file.
FCH$M_DIRECTORY	File is a directory.
FCH$M_BADBLOCK	File contains bad blocks.
FCH$M_MARKDEL	File is marked for deletion.
FCH$M_ERASE	Erase file contents before deletion.

1.4 ACP QIO Record Attributes Area

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

Figure 1–7 ACP–QIO Record Attributes Area


31
24 23
16 15
8 7
0
FAT$W_RSIZE
FAT$B_RATTRIB
FAT$B_RTYPE *
FAT$L_HIBLK
4
FAT$L_EFBLK
8
FAT$B_VFCSIZE
FAT$B_BKTSIZE
FAT$W_FFBYTE
12
FAT$W_DEFEXT
FAT$W_MAXREC
16
FAT$W_GBC
20
(6 bytes reserved for future use)
24
FAT$W_VERSIONS
not used
28

*FAT$V_RTYPE bits 0-3; FAT$V_FILEORG bits 4-7

ZK-641-82

Table 1–8 lists the record attributes values and their meanings.

Table 1–8 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
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$M_NOSPAN	No spanned records
FAT$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.	

[1] Defined but not implemented.

[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.

Table 1–8 (Cont.) ACP Record Attributes Values

Field Value	Meaning
FAT$W_MAXREC	Maximum record size in bytes.
FAT$W_DEFEXT	Default extend quantity.
FAT$W_GBC	Global buffer count.
FAT$W_VERSIONS	Default version limit; valid only if the file is a directory.

1.5 ACP–QIO Attributes Statistics Block

Figure 1–8 shows the format of the attributes statistics block. Table 1–9 lists the contents of this block.

Figure 1–8 ACP–QIO Attributes Statistics Block

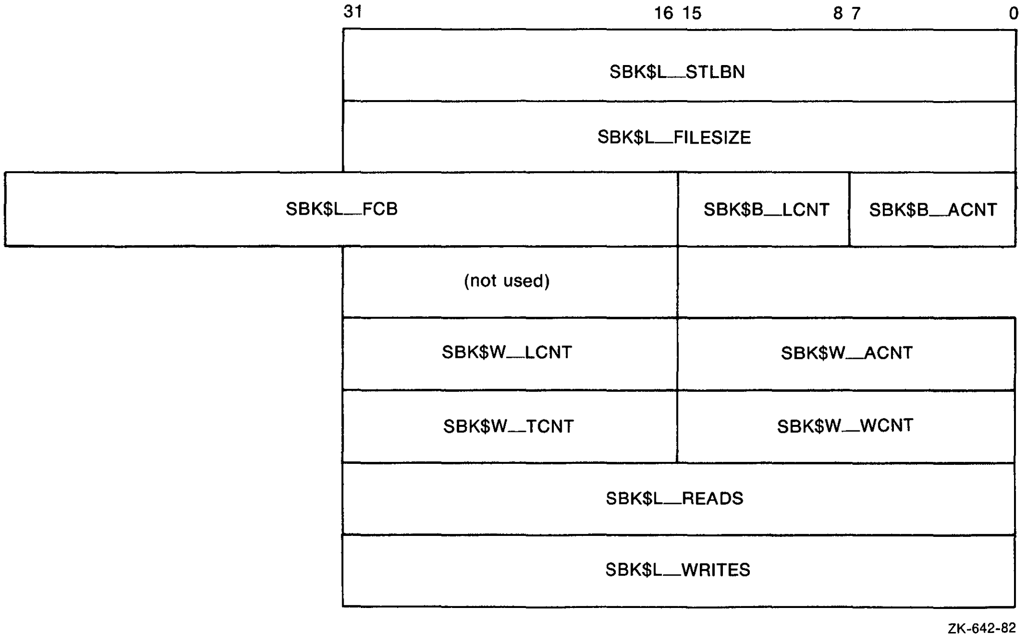

31
16 15
8 7
0
SBK$L_STLBN
SBK$L_FILESIZE
SBK$L_FCB
SBK$B_LCNT
SBK$B_ACNT
(not used)
SBK$W_LCNT
SBK$W_ACNT
SBK$W_TCNT
SBK$W_WCNT
SBK$L_READS
SBK$L_WRITES
ZK-642-82

Table 1–9 Contents of the Statistics Block

Field	Field Values	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, that is, the high- and low-order 16 bits are transposed for PDP–11 compatibility. The following subfields are defined:

Table 1–9 (Cont.) Contents of the Statistics Block

Field	Field Values	Meaning
	SBK$W_STLBNH	Starting LBN (high-order 16 bits).
	SBK$W_STLBNL	Starting LBN (low-order 16 bits).
SBK$L_FILESIZE		Contains the size of the file in blocks. The file size appears as an inverted longword, that is, the high- and low-order 16 bits are transposed, for PDP–11 compatibility. The following subfields are defined:
	SBK$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). This field is present for PDP–11 compatibility.
SBK$B_LCNT[1]		Lock count (low byte). This field is present for PDP–11 compatibility.
SBK$L_FCB		System pool address of the file's file control block.
SBK$W_ACNT[1]		Access count, that is, the number of channels that currently have the file open.
SBK$W_LCNT[1]		Lock count, that is, the number of access operations that have locked the file against writers.
SBK$W_WCNT[1]		Writer count, that is, the number of channels that currently have the file open for write.
SBK$W_TCNT[1]		Truncate lock count, that is, the number of access operations that have locked the file against truncation.
SBK$L_READS		The number of read operations executed for the file on this channel.
SBK$L_WRITES		The number of write operations executed for the file on this channel.

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

1.6 Major Functions

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

1.6.1 Create File

This virtual I/O function creates a directory entry and/or a file on a disk device, or a file on a magnetic tape device.

The function code is:

- IO$_CREATE

The function modifiers are:

- IO$M_CREATE—creates a file.
- IO$M_ACCESS—opens the file on the user's channel.
- IO$M_DELETE—marks the file for deletion (applicable only to disk devices).

1.6.1.1 Input Parameters

The device/function-dependent arguments for IO$_CREATE are:

- P1—the address of the file information block (FIB) descriptor.
- P2—the address of the file name string descriptor (optional).
- P3—the address of the word that is to receive the length of the resultant file name string (optional).
- P4—the address of a descriptor for a buffer that is to receive the resultant file name string (optional).
- P5—the address of a list of attribute descriptors (optional).

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

Field	Field Values	Meaning
FIB$L_ACCTL		Specifies field values that control access to the file. The following bits are applicable to the IO$_CREATE function:
	FIB$M_REWIND	Set to rewind magnetic tape before creating the file. Any data currently on the tape will be overwritten.
	FIB$M_CURPOS	Set to create magnetic tape file at the current tape position. (Note: a magnetic tape file will be created at the end of the volume set if neither FIB$M_REWIND nor FIB$M_CURPOS is set). If the tape is not positioned at the end of a file, FIB$M_CURPOS creates a file at the next file position. Any data currently on the tape past the current file position will be overwritten.

Field	Field Values	Meaning
	FIB$M_WRITETHRU	Specifies that the file header is to be written back to the disk. If not specified and the file is opened, writing of the file header may be deferred to some later time.
FIB$W_CNTRLFUNC		Specifies the following value that allows the user to control actions subsequent to EOT detection on a magnetic tape file.
	FIB$C_USEREOT	Set on a per file basis to specify user EOT mode. If this bit is set, the magnetic tape driver notifies the magnetic tape system if EOT has been detected (considered a 'serious exception') during the creation of a file. The magnetic tape system, in turn, returns the alternate success code SS$_ENDOFTAPE or SS$_ENDOFVOLUME to the user. All subsequent I/O requests are completed with a failure status return of SS$_SERIOUSEXP. The driver will not execute any I/O functions until the serious exception has been explicitly cleared by issuing an IO$_ACPCONTROL function (see Section 1.6.7). If the file is deaccessed or closed, the user EOT mode is cleared on further processing of the magnetic tape.
FIB$W_FID		Contains the file ID of the file created or entered.
FIB$W_DID		Contains the file identifier of the directory file.
FIB$W_NMCTL		Controls the processing of the file name in a directory operation. The following bits are applicable to the IO$_CREATE function:
	FIB$M_NEWVER	Set to create file of same name with next higher version number. Only for disk devices.
	FIB$M_SUPERSEDE	Set to supersede an existing file of the same name, type, and version. Only for disk devices.

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

1.6.1.2 Disk ACP Operation

If the modifier IO$M_CREATE is specified, a file is created. The file ID of the file created is returned in FIB$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.) However, 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$M_NEWVER is set, the directory entry created gets a version number one greater than the highest previously existing version of that file, or the specified version number, whichever is greater.
- If the version number in the specified file name is a nonzero number and the directory already contains a file of the same name, type, and version, the previously existing file is set aside for deletion if FIB$M_SUPERSEDE is specified. If FIB$M_SUPERSEDE is not specified, the create operation will fail with an SS$_DUPFILNAM status.
- If, after creating the new directory entry, the number of versions of the file exceed 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; otherwise it is taken from the default version limit of the directory file; if that 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 they are present. The file name string is also written into the header. If no directory operation was requested, that is, 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$M_EXTEND is specified in the FIB, an extend subfunction is performed (see Section 1.3.3).

Finally, if a file was set aside for deletion by the directory operation described previously, that file is deleted. If the file is deleted because the FIB$M_SUPERSEDE bit was set, the alternate success status SS$_SUPERSEDE is returned in the I/O status block. If the file is deleted because the version limit was exceeded, the alternate success status SS$_FILEPURGED is returned.

If an error occurs in the operation of an IO$_CREATE function, any actions performed so far 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 Magnetic Tape ACP Operation

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

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 function code is:

- IO$_ACCESS

The function modifiers are:

- IO$M_CREATE—creates a file.
- IO$M_ACCESS—opens the file on the user's channel.

1.6.2.1 Input Parameters

The device/function-dependent arguments for IO$_ACCESS are:

- P1—the address of the file information block (FIB) descriptor.
- P2—the address of the file name string descriptor (optional).
- 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$_ACCESS operation:

Field	Field Values	Meaning
FIB$W_CNTRLFUNC		Specifies the following value that allows the user to control actions subsequent to EOT detection on a magnetic tape file.
	FIB$C_USEREOT	Set on a per file basis to specify user EOT mode. If this bit is set, the magnetic tape driver notifies the magnetic tape system when EOT has been detected (considered a 'serious exception') when a file is accessed. The magnetic tape system, in turn, returns the alternate success code SS$_ENDOFTAPE or SS$_ENDOFVOLUME to the user. All subsequent I/O requests are completed with a failure status return of SS$_SERIOUSEXP. The driver will not execute any I/O functions until the serious exception has been explicitly cleared by issuing an IO$_ACPCONTROL function (see Section 1.6.7). If the file is deaccessed or closed, the user EOT mode is cleared on further processing of the magnetic tape.

Field	Field Values	Meaning
FIB$W_VERLIMIT		Receives the version limit for the file. Applicable only if FIB$W_DID is a nonzero number, that is, 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.
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 is deleteable. 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$M_CREATE function modifier is specified, the function is then reexecuted as a CREATE. In that case, the argument interpretations for IO$_CREATE apply, rather than those for IO$_ACCESS.

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

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

The function code is:

- IO$_DEACCESS

IO$_DEACCESS takes no function modifiers.

1.6.3.1 Input Parameters

The device/function-dependent arguments for IO$_DEACCESS are:

- 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	Field Values	Meaning
FIB$W_FID		File identification of the file being deaccessed. This field may contain a value of 0. If it does not, it must match the file identifier of the accessed file.
FIB$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.

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 and no attributes are specified, and FIB$M_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$M_TRUNCATE is specified in the FIB, a truncate subfunction is performed (see Section 1.3.4).

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

1.6.4 Modify File

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

The function code is:

- IO$_MODIFY

IO$_MODIFY takes no function modifiers.

1.6.4.1 Input Parameters

The device/function-dependent arguments for IO$_MODIFY are:

- P1—the address of the file information block (FIB) descriptor.
- P2—the address of the file name string descriptor (optional). If specified, the 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	Field Values	Meaning
FIB$L_ACCTL		Specifies field values that control access to the file. The following bits are applicable to the IO$_MODIFY function:
	FIB$M_WRITETHRU	Specifies that the file header is to be written back to the disk. If not specified and the file is currently open, writing of the file header may be deferred to some later time.
FIB$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 may be either extended or truncated. If FIB$M_EXTEND is specified in the FIB, an extend subfunction is performed (see Section 1.3.3). If FIB$M_TRUNCATE is specified in the FIB, a truncate subfunction is performed (see Section 1.3.4). Users should note that extend and truncate operations are mutually exclusive.

1.6.5 Delete File

This virtual I/O function removes a directory entry and/or file header from a disk volume.

The function code is:

- IO$_DELETE

The function modifier is:

- IO$M_DELETE—deletes the file (or marks it for deletion).

The device/function-dependent arguments for IO$_DELETE are:

- P1—the address of the file information block (FIB) descriptor.
- P2—the address of the file name string descriptor (optional).
- P3—the address of the word that is to receive the length of the resultant file name string (optional).
- P4—the address of a descriptor for a buffer that is to receive the resultant file name string (optional).

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

Field	Field Values	Meaning
FIB$L_ACCTL		Specifies field values that control access to the file. The following bit is applicable to the IO$_DELETE function:
	FIB$M_WRITETHRU	Specifies that the file header is to be written back to the disk. If not specified and the file is currently open, writing of the file header may be deferred to some later time.
FIB$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 Mount

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

1.6.7 ACP Control

This virtual I/O function performs miscellaneous control functions, depending on the arguments specified.

The function code is:

- IO$_ACPCONTROL

The function modifier is:

- IO$M_DMOUNT—dismounts a volume.

1.6.7.1 Input Parameters

The device/function-dependent arguments for IO$_ACPCONTROL are:

- P1—the address of the file information block (FIB) descriptor.
- P2—the address of the file name string descriptor (optional).
- P3—the address of the word that is to receive the length of the resultant file name string (optional).
- P4—the address of a descriptor for a buffer that is to receive the resultant file name string (optional).

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

Field	Field Values	Meaning
FIB$W_CNTRLFUNC		Specifies the control function to be performed. This field overlays FIB$W_EXCTL.
FIB$L_CNTRLVAL		Specifies additional function-dependent data. This field overlays FIB$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.
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.

Field	Field Values	Meaning
	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 is deleteable. The mask has the same configuration as the standard protection mask.

1.6.7.2 Magnetic Tape Control Functions

The following FIB field is applicable to magnetic tape operations:

Field	Field Values	Meaning
FIB$W_CNTRLFUNC		Several ACP control functions are used for magnetic tape positioning. These functions are specified by supplying a FIB with P1 containing the FIB descriptor address. Modifiers and parameters P2, P3, and P4 are not allowed. These functions clear serious exceptions in magnetic tape drivers. The following control functions may be specified to control magnetic tape positioning:
	FIB$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.

Field	Field Values	Meaning
	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 Sections 1.6.1 and 1.6.2). This allows the user to write data blocks beyond the EOT marker, which may result in the magnetic tape not conforming to the ANSI standard for magnetic tapes (see ANSI Standard X3.27 - 1978).

1.6.7.3 Miscellaneous Disk Control Functions

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

IO$M_DMOUNT	Specifying the dismount modifier on the IO$_ACPCNTRL function executes a dismount QIO. No parameters in the FIB are used; in fact, the FIB may be omitted. This function does not actually 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, the user 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, the user must have either a system UIC or SYSPRV privilege, or be the owner of the volume.

1.6.7.4 Disk Quotas

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

Table 1–10 lists the enable and disable quota control functions.

Table 1–10 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, the user must have either a system UIC or SYSPRV privilege, or be 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, the user must have either a system UIC or SYSPRV privilege, or be owner of the volume.

Table 1–11 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.

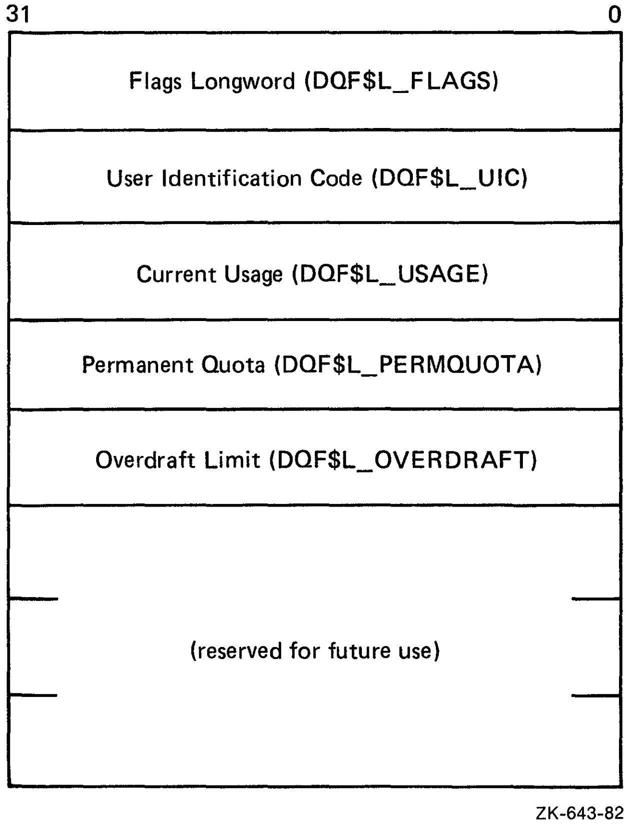
IO$_ACPCONTROL functions that transfer quota file data between the caller and the ACP use the following device/function-dependent arguments:

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

Table 1–11 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$M_ALL_MEM Match all UIC members FIB$M_ALL_GRP Match all UIC groups The ACP maintains position context in FIB$L_WCC. On the first examine call, the user should specify 0 in FIB$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 entries not belonging to the user.
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$M_MOD_PERM Change the permanent quota FIB$M_MOD_OVER Change the overdraft quota FIB$M_MOD_USE Change the usage data The usage data can be changed only if the volume is locked by FIB$C_LOCK_VOL (see Section 1.6.7.3). FIB$C_MOD_QUOTA requires write access to the quota file. The P3 and P4 arguments return the modified quota entry to the user. By using the flags FIB$M_ALLMEM and FIB$M_ALLGRP, you can search through the quota file using wildcards just as you would with the FIB$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 the user. By using the flags FIB$M_ALLMEM and FIB$M_ALLGRP, 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


31
0
Flags Longword (DQF$L_FLAGS)
User Identification Code (DQF$L_UIC)
Current Usage (DQF$L_USAGE)
Permanent Quota (DQF$L_PERMQUOTA)
Overdraft Limit (DQF$L_OVERDRAFT)
(reserved for future use)
ZK-643-82

1.7 I/O Status Block

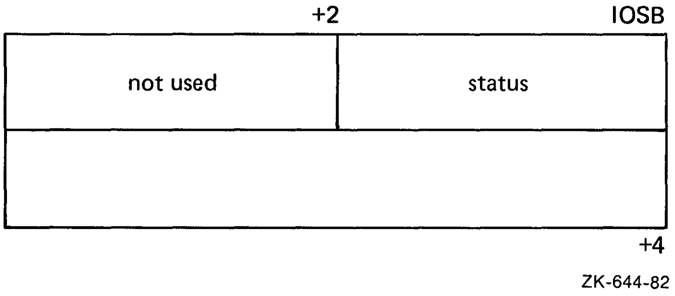
Figure 1–10 shows the I/O status block (IOSB) for ACP–QIO functions. Appendix A lists the status returns for these functions. (The *VAX/VMS System Messages and Recovery Procedures Reference Manual* provides explanations and suggested user actions for these returns.)

The file ACP returns a completion status in the first longword of the IOSB. In an extend operation, the second longword is used to return the number of blocks allocated to the file. If a contiguous extend operation (FIB$M_ALCON) fails, the second longword is used to return the size of the file after truncation.

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

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.

Figure 1–10 IOSB Contents - ACP–QIO Functions


+2
IOSB
not used
status
+4
ZK-644-82

2 Card Reader Driver

This section describes the use of the VAX/VMS card reader driver that supports the CR11 card reader.

2.1 Supported Card Reader Device

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

2.2 Driver Features and Capabilities

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

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

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

2.2.1 Read Modes

The VAX/VMS system provides two card reader device/function-dependent modifier bits for read data operations:

- IO$M_PACKED—read packed Hollerith code
- IO$M_BINARY—read binary code

If IO$M_PACKED is set, the data is packed and stored in sequential bytes of the input buffer. If IO$M_BINARY is set, the data is read and stored in sequential words of the input buffer. IO$M_BINARY takes precedence over IO$M_PACKED.

The read mode can also be set by a special card punch combination that sets the translation mode (see Section 2.2.2.2) or by the set mode function (see Section 2.4.3).

2.2.2 Special Card Punch Combinations

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

2.2.2.1 End-of-File Condition

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

2.2.2.2 Set Translation Mode

If the read mode is nonbinary, nonpacked Hollerith (the IO$M_BINARY and IO$M_PACKED function modifiers are not set), the current translation mode can be set to the 026 or 029 punch code. (Table 2–5 lists the 026 and 029 punch codes.) A card with the 12-2-4-8 holes punched in column 1 sets the translation mode to the 026 code. A card with the 12-0-2-4-6-8 holes punched in column 1 sets the translation mode to the 029 code. The translation mode can be changed as often as required.

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

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

2.2.3 Error Recovery

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

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

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

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

- Pick check—The next card cannot be delivered from the input hopper to the read mechanism. To recover from this error, remove the next card to be read from the input hopper and smooth the leading edge, that is, the edge that will enter the read mechanism first. Replace the card in the input hopper and press the RESET button. The card reader operation will resume automatically. If a pick check error occurs again on the same card, remove the card from the input hopper and repunch it. Place the duplicate card in the input hopper and press the RESET button. If the problem persists, either an adjustment is required or nonstandard cards are in the input hopper.

- Stack check—The card just read did not stack properly in the output hopper. To recover from this error, remove the last card read from the output hopper and examine its condition. If it is excessively worn or mutilated, repunch it. Place either the duplicate card or the original card in the read station of the input hopper and press the RESET button. The card reader operation will resume automatically. If the stack check error recurs immediately, an adjustment is required.
- Hopper check—Either the input hopper is empty or the output hopper is full. To recover from this error, examine the input hopper and, if empty, either load the next deck of input cards or an end-of-file card. If the input hopper is not empty, remove the cards that have accumulated in the output hopper and press the RESET button. The card reader operation will resume automatically.
- Read check—The last card was read incorrectly. To recover from this error, remove the last card from the output hopper and examine its condition. If it is excessively worn, mutilated, or contains punches before column 0 or after column 80, repunch the card to correct any incorrect punches. Place either the original card or duplicate card in the read station of the input hopper and press the RESET button. The card reader operation will resume automatically. If the read check error recurs immediately, an adjustment is necessary.

2.3 Device Information

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

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

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

Table 2–1 Card Reader Device-Independent Characteristics

Characteristic[1]	Meaning
Dynamic Bit (Conditionally Set)	
DEV$M_AVL	Device is online and available.
Static Bits (Always Set)	
DEV$M_IDV	Device is capable of input.
DEV$M_REC	Device is record-oriented.

[1] Defined by the $DEVDEF macro.

Table 2–2 Device-Dependent Characteristics for Card Readers

Value[1]	Meaning
CR$V_TMODE CR$S_TMODE	Specifies the translation mode for nonbinary, nonpacked Hollerith data transfers.[2] Possible values are: CR$K_T026 Translate according to 026 punch code CR$K_T029 Translate according to 029 punch code

[1] Defined by the $CRDEF macro.

[2] Section 2.2.2.2 describes the set translation mode punch code.

2.4 Card Reader Function Codes

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

Table 2–3 Card Reader I/O Functions

Function Code and Arguments	Type[1]	Function Modifiers	Function
IO$_READLBLK P1,P2	L	IO$M_BINARY IO$M_PACKED	Read logical block.
IO$_READVBLK P1,P2	V	IO$M_BINARY IO$M_PACKED	Read virtual block.
IO$_READPBLK P1,P2	P	IO$M_BINARY IO$M_PACKED	Read physical block.
IO$_SENSEMODE	L		Sense the card reader characteristics and return them in the I/O status block.
IO$_SETMODE P1	L		Set card reader characteristics for subsequent operations.
IO$_SETCHAR P1	P		Set card reader characteristics for subsequent operations.

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

2.4.1 Read

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

The VAX/VMS system provides three read function codes:

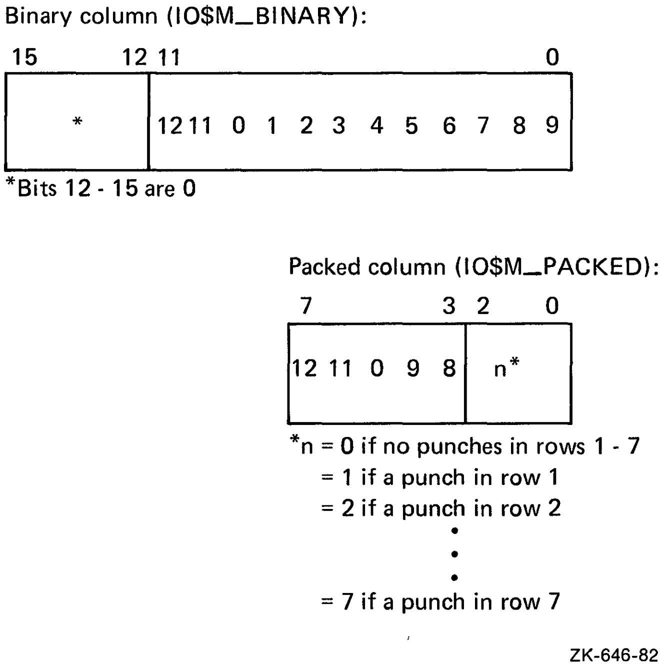
- IO$_READVBLK—read virtual block
- IO$_READLBLK—read logical block
- IO$_READPBLK—read physical block

Two function-dependent arguments are used with these codes:

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

The read binary function modifier (IO$M_BINARY) and the read packed Hollerith function modifier (IO$M_PACKED) can be used with all read functions. If IO$M_BINARY is specified, successive columns of data are stored in sequential word locations of the input buffer. If IO$M_PACKED is specified, successive columns of data are packed and stored in sequential byte locations of the input buffer. If neither of these function modifiers is specified, successive columns of data are translated in the current mode (026 or 029) and are stored in sequential bytes of the input buffer. Figure 2–1 shows how data is stored by IO$M_BINARY and IO$M_PACKED.

Figure 2–1 Binary and Packed Column Storage


Binary column (IO$M_BINARY):
15
12 11
0
*
12 11 0 1 2 3 4 5 6 7 8 9
*Bits 12 - 15 are 0
Packed column (IO$M_PACKED):
7
3 2
0
12 11 0 9 8
n*
*n = 0 if no punches in rows 1 - 7
= 1 if a punch in row 1
= 2 if a punch in row 2
= 7 if a punch in row 7
ZK-646-82

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

2.4.2 Sense Card Reader Mode

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

2.4.3 Set Mode

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

- Set mode
- Set characteristic

2.4.3.1 Set Mode

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

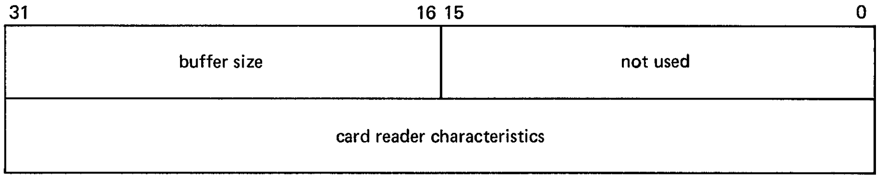
- IO$_SETMODE

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

- P1—the address of a characteristics buffer

Figure 2–2 shows the quadword set mode characteristics buffer.

Figure 2–2 Set Mode Characteristics Buffer



ZK-647-82

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

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

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

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

Table 2–5 Card Reader Codes

Character	$ASCII_8$	DEC029	DEC026
{	173	12 0	12 0
}	175	11 0	11 0
SPACE	40	NONE	NONE
!	41	11 8 2	12 8 7
"	42	8 7	0 8 5
_	43	8 3	0 8 6
$	44	11 8 3	11 8 3
%	45	0 8 4	0 8 7
&	46	12	11 8 7
'	47	8 5	8 6
(	50	12 8 5	0 8 4
)	51	11 8 5	12 8 4
*	52	11 8 4	11 8 4
+	53	12 8 6	12
,	54	0 8 3	0 8 3
-	55	11	11
.	56	12 8 3	12 8 3
/	57	0 1	0 1
0	60	0	0
1	61	1	1
2	62	2	2
3	63	3	3
4	64	4	4
5	65	5	5
6	66	6	6
7	67	7	7
8	70	8	8
9	71	9	9
:	72	8 2	11 8 2
;	73	11 8 6	0 8 2
<	74	12 8 4	12 8 6
=	75	8 6	8 3
>	76	0 8 6	11 8 6
?	77	0 8 7	12 8 2
@	100	8 4	8 4
A	101	12 1	12 1
B	102	12 2	12 2
C	103	12 3	12 3
D	104	12 4	12 4

Table 2–5 (Cont.) Card Reader Codes

Character	$ASCII_8$	DEC029	DEC026
E	105	12 5	12 5
F	106	12 6	12 6
G	107	12 7	12 7
H	110	12 8	12 8
I	111	12 9	12 9
J	112	11 1	11 1
K	113	11 2	11 2
L	114	11 3	11 3
M	115	11 4	11 4
N	116	11 5	11 5
O	117	11 6	11 6
P	120	11 7	11 7
Q	121	11 8	11 8
R	122	11 9	11 9
S	123	0 2	0 2
T	124	0 3	0 3
U	125	0 4	0 4
V	126	0 5	0 5
W	127	0 6	0 6
X	130	0 7	0 7
Y	131	0 8	0 8
Z	132	0 9	0 9
[	133	12 8 2	11 8 5
\	134	11 8 7	8 7
]	135	0 8 2	12 8 5
↑ or ^	136	12 8 7	8 5
← or _	137	0 8 5	8 2

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

2.4.3.2 Set Characteristic

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

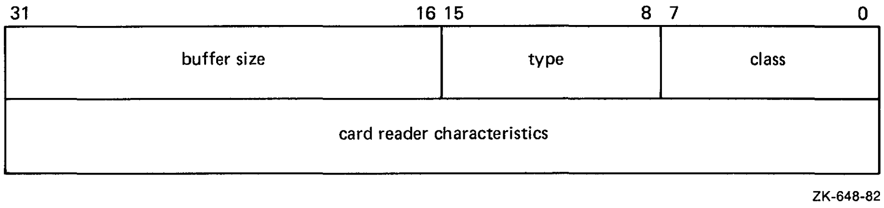
- IO$_SETCHAR

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

- P1—the address of a characteristics buffer

Figure 2–3 shows the set characteristic characteristics buffer.

Figure 2–3 Set Characteristic Buffer

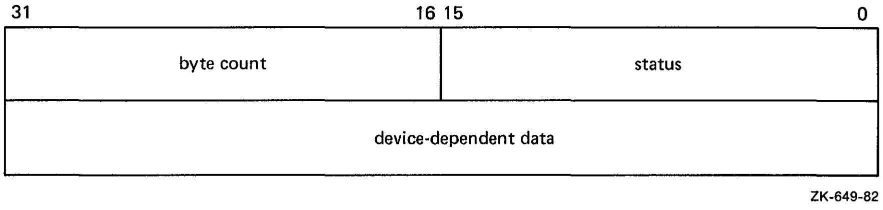

31
16 15
8 7
0
buffer size
type
class
card reader characteristics
ZK-648-82

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

2.5 I/O Status Block

The I/O status block (IOSB) format for QIO functions on the card reader is shown in Figure 2–4. Appendix A lists the status returns for these functions. (The *VAX/VMS System Messages and Recovery Procedures Reference Manual* provides explanations and suggested user actions for these returns.) Table 2–2 lists the device-dependent data returned in the second longword. The IO$_SENSEMODE function can be used to obtain this data.

Figure 2–4 IOSB Contents


31
16 15
0
byte count
status
device-dependent data
ZK-649-82

3 Disk Drivers

This section describes the use of VAX/VMS disk drivers. These drivers support the devices listed in Table 3–1 and discussed in Section 3.1.

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

3.1 Supported Disk Devices and Controllers

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

Table 3–1 Supported Disk Devices

Model	Code	Type	DSA	Logical Blocks/Drive
RA60	DJ	Packed	Yes	400,176
RA80	DU	Fixed	Yes	236,964
RA81	DU	Fixed	Yes	891,072
RB02	DQ	Cartridge	No	20,480
RB80	DQ	Fixed	No	242,606
RC25	DA	Fixed, Cartridge	Yes[1]	102,400[2]
RRD50	DU	Optical	Yes[1]	1,669,400
RD51	DU	Fixed	Yes[1]	21,600
RD52	DU	Fixed	Yes[1]	60,480
RD53	DU	Fixed	Yes[1]	138,672
RL02	DL	Cartridge	No	20,480
RM03	DR	Packed	No	131,680
RM05	DR	Packed	No	500,384
RM80	DR	Fixed	No	242,606
RP05	DB	Packed	No	171,798
RP06	DB	Packed	No	340,670

[1] Incompatible with the UDA50, KDA50 and KDB50.

[2] 51,200 fixed; 51,200 cartridge.

Table 3–1 (Cont.) Supported Disk Devices

Model	Code	Type	DSA	Logical Blocks/Drive
RP07	DR	Fixed	No	1,008,000
RK06	DM	Cartridge	No	27,126
RK07	DM	Cartridge	No	53,790
RX01	DX	Flexible	No	494
RX02	DY	Flexible	No	494[3] 988[4]
RX50	DU	Flexible	Yes[1]	800
TU58[5]	DD	Cartridge	No	512

[1] Incompatible with the UDA50, KDA50 and KDB50.

[3] Single density (See Section 3.3).

[4] Double density (See Section 3.3).

[5] A magnetic tape device, the TU58 operationally resembles a disk device. See Section 3.1.13 for a description of the TU58 in disk terms.

3.1.1 UDA50 UNIBUS Disk Adapter

The UDA50 UNIBUS Disk Adapter (UDA50) is a microprocessor-based disk controller for mass storage devices that implement the DIGITAL Storage Architecture (DSA); for more information on the DSA, see Section 3.2.6.

The UDA50 is used to connect any combination of four RA60, RA80, and RA81 disk drives to the UNIBUS. Two UDA50 controllers can be attached to a single UNIBUS for a maximum of eight disk drives per UNIBUS. (On the VAX–11/780 processor, VAX/VMS supports one UDA50 on the first UNIBUS, to which can be added certain other options. Adding a second UDA50 requires a second UNIBUS. With the exception of the first UNIBUS, a maximum of two UDA50s per UNIBUS are supported. If two UDA50s are on a UNIBUS, then 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 VAX/VMS operating system processes request virtual or logical I/O on disks controlled by the UDA50. VAX/VMS maps virtual block addresses into logical block addresses. The UDA50 then resolves logical block addresses into physical block addresses on the disk.

The UDA50 corrects bad blocks on the disk by requesting that the disk class driver revector a failing physical block to another, error-free physical block on the disk; the logical block number is not changed. The VAX/VMS system, which does logical or virtual I/O to such a disk, is never aware that any bad blocks might exist on a disk attached to an UDA50. The UDA50 also corrects most data errors.

3.1.2 KDA50 Disk Controller

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

3.1.3 KDB50 Disk Controller

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

3.1.4 HSC50 Controller

The HSC50 is a high-speed, high-availability controller for mass storage devices that implement the DIGITAL Storage Architecture (DSA); for more information on the DSA, see Section 3.2.6. The HSC50 is connected to a processor by way of a Computer Interconnect (CI). The VAX/VMS operating system supports the use of the HSC50 in controlling the RA60, RA80, and RA81 disks.

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

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

The HSC50 provides access to disks despite most hardware failures, and allows 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 VAX/VMS operating system allows read/write access to all nodes that are members of the same VAXcluster.**

The HSC50 allows the user to add or subtract disks from the device configuration without rebooting the system.

3.1.5 RA60 Pack Disk

The RA60 is a large-capacity, removable disk that provides 205 MB of usable storage (7.5 million bits of data per square inch) with transfer rates of 1.9 MB/second (burst) and 950 KB/second (sustained). The RA60 belongs to the DIGITAL Storage Architecture (DSA) family of disk devices (see Section 3.2.6). It is connected to either a UNIBUS Disk Adapter (UDA50) or an HSC50 controller. Up to 4 disk drives can be connected to each UDA50. Up to 24 disk drives can be connected to each HSC50.

3.1.6 RB02 and RL02 Cartridge Disks

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

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

3.1.7 RM03 and RM05 Pack Disks

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

3.1.8 RA80/R80/RM80 and RA81 Fixed Media Disks

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

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

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

The RA80 and RA81 belong to the DIGITAL Storage Architecture (DSA) family of disk devices (see Section 3.2.6).

3.1.9 RP05 and RP06 Pack Disks

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

3.1.10 RP07 Fixed Media Disk

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

3.1.11 RK06 and RK07 Cartridge Disks

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

3.1.12 RC25 Disk

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

3.1.13 RRD50 Read-Only Memory (CDROM)

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

3.1.14 RX01 Console Disk

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

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

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

3.1.15 RX02 Floppy Disk

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

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

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

For physical block I/O operations, the track and sector parameters shown in Figure 3–2 describe a physical sector (128 bytes in single density; 256 bytes in double density). Note that the driver does not apply track-to-track skew, cylinder offset, or sector interleaving to the physical medium address.

3.1.16 TU58 Magnetic Tape (DECtape II)

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

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

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

3.2 Driver Features and Capabilities

VAX/VMS disk drivers provide the following capabilities:

- 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)

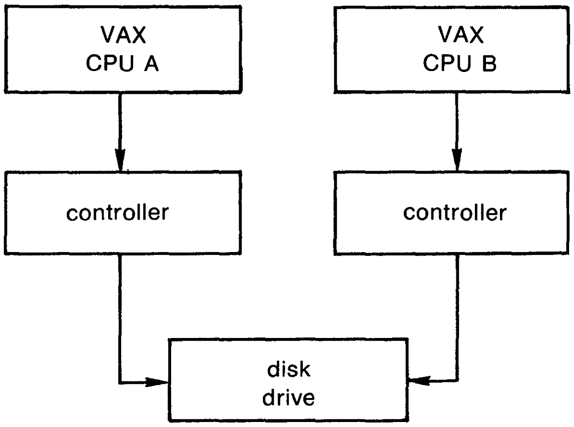
- Data checks on a per-request, per-file, and/or per-volume basis (except RX01 and RX02)
- Full recovery from power failure for online disk drives with volumes mounted
- Extensive error recovery algorithms, for example, error code correction and offset (except RB02, RL02, RX01, RX02, and TU58); for DSA 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 the data check, overlapped seek, error recovery, and logical-to-physical translation capabilities in greater detail.

3.2.1 Dual Porting (MASSBUS)

The VAX/VMS MASSBUS disk drivers, DBDRIVER and DRDRIVER, support static dual porting. Dual porting allows two MASSBUS controllers to access the same disk drive. Figure 3–1 shows this configuration. The RP05, RP06, RP07, RM03, RM05, and RM80 disk drives can be ordered, or upgraded in the field, with the MASSBUS dual port option.

Figure 3–1 Dual-Ported Disk Drives



ZK-650-82

3.2.1.1 Port Selection and Access Modes

The port select switch(es), on each disk drive, selects the port(s) from which the drive can be accessed. A drive can be in one of three access modes:

- Locked on Port A—the drive is in a single-port mode (Port A). It will not respond to any request on Port B.

- Locked on Port B—the drive is in a single-port mode (Port B). It will 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 three 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 one second timeout occurs. After the connected port is released from its controller, the drive checks the other port's request flag to determine whether there has been a request on that port. If there is no request pending, the drive returns to the neutral state.

3.2.1.2 Disk Use and Restrictions

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

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

See the *Guide to VAXclusters* for additional information on dual-ported disk operation.

3.2.1.3 Dual-Porting DSA Disks

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

3.2.2 Data Check

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

Disk drivers support data checks at three levels:

- Per request—Users can specify the data check function modifier (IO$M_DATACHECK) on a read logical block, write logical block, read virtual block, write virtual block, read physical block, or write physical block operation. IO$M_DATACHECK is not supported for the RX01 and RX01 drivers.
- Per volume—Users can specify the characteristics "data check all reads" and/or "data check all writes" when the volume is mounted. The *VAX/VMS DCL Dictionary* describes volume mounting and dismounting. The *VAX/VMS System Services Reference Manual* in the *VAX/VMS System Routines Reference Volume* describes the Mount Volume ($MOUNT) and Dismount Volume ($DISMOU) system services.
- Per file—Users can specify the file access attributes "data check on read" or "data check on write." File access attributes are specified when the file is accessed. Section 1 of this manual and the *VAX Record Management Services Reference Manual* in the *VAX/VMS System Routines Reference Volume* describe file access.

Offset recovery is performed during a data check but Error Code Correctable (ECC) correction is not performed (see Section 3.2.4). For example, if a read operation is performed and an ECC correction applied, the data check would fail even though the data in memory is correct. In this case, the driver returns a status code indicating that the operation was successfully completed, but the data check could not be performed because of an ECC correction.

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

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

3.2.3 Overlapped Seeks

A seek operation involves moving the disk read/write heads to a specific place on the disk without any transfer of data. All transfer functions, including data checks, are preceded by an implicit seek operation (except when the seek is inhibited by the physical I/O function modifier IO$M_INHSEEK). Except on RL02, RX01, RX02, and TU58 drives, seek operations can be overlapped. That is, when one drive performs a seek operation, any number of other drives can also perform seek operations.

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

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

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

3.2.4 Error Recovery

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

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

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 reexecuted up to eight times before a fatal error is returned.

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

3.2.4.1 Skip Sectoring

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

Bad blocks may be detected 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 at the user interface. The device appears to have 31 physically contiguous sectors per track. Sector 32 is not directly addressable, although it will be accessed if a skip sector is present on the track.

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

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

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

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

   ```
   Number of sectors per track = 26
   Number of tracks per cylinder = 1
   Number of cylinders per disk = 77
   ```

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

   ```
   Interleaving:
   ITEMP = ISECT*2
   IF (ISECT .GT. 12) ITEMP = ITEMP-25
   ISECT = ITEMP
   Skew:
   ISECT = ISECT+(6*ICYL)
   ISECT = MOD (ISECT, 26)
   ```

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

   ```
   ISECT = ISECT+1
   ```

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

   ```
   ICYL = ICYL+1
   ```

3.2.6 DIGITAL Storage Architecture (DSA) Devices

The DIGITAL Storage Architecture (DSA) is a collection of specifications that cover all aspects of a mass storage product. The specifications are grouped into three general categories: media format, drive-to-controller interconnect, and controller-to-host communications.

The media format specifications describe the structure of sectors on a disk and the algorithms for replacing bad blocks. The drive-to-controller specifications describe the connection between an RA60, RA80, or RA81 drive and its controller. The controller-to-host specifications describe how hosts request controllers to transfer data.

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

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 ANALYZE /DISK_STRUCTURE /READ_CHECK, may be used to check the integrity of newly received disks.) See Section 3.2.6.1 for additional information on bad block replacement for DSA disks.
- You must insert a WAIT statement in your SYSTARTUP.COM file prior to the first MOUNT statement for a DSA disk. The wait period should be about two to four seconds for the UDA50 and about 30 seconds for the HSC50. The wait time is controller-dependent and can be as much as several minutes if the controller is offline or otherwise inoperative. If this wait is omitted, the MOUNT request may fail with a "no such device" status.
- The DUDRIVER and the DSA device controllers allow multiple, concurrently outstanding QIO requests. The order in which these requests complete may not be in the order in which they were issued.
- All DSA disks can be dual-ported, but only one HSC/UDA controller can control a disk at a time (see Section 3.2.1.3). DSA disks that use an internal controller cannot be dual-ported.
- In many cases you can attach a DSA disk to its controller on a running VAX/VMS system and then use it immediately without manual intervention.
- DSA disks and the DUDRIVER do not accept physical QIO data transfers or seek operations.

3.2.6.1 Bad Block Replacement and Forced Errors for DSA Disks

Disks that are built according to the DSA specifications appear to be error free. Some number of logical blocks are always good, that is, 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 to one of the replacement blocks. The logical block is described as being *revectored*. The process that revectors logical blocks is called a *badblock replacement* operation. Bad block replacement operations use data stored in a special area of the disk called the *Replacement and Caching Table (RCT)*.

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

The design goal of DSA disks is that this read proceed without error and that the RCT copy of the data be 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 translates into 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 VAX/VMS utilities and DCL commands treat the forced error flag as a fatal error and will 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 in question and act accordingly.

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

3.3 Device Information

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

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

Table 3–2 Disk Device Characteristics

Characteristic[1]	Meaning
Dynamic Bits (Conditionally Set)	
DEV$M_AVL	Device is online and available.
DEV$M_CDP[2]	Dual-path device with two 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$M_SRV[2]	For a VAXcluster, device is served by the MSCP server.
Static Bits (Always Set)	
DEV$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.

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

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

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

3.4 Disk Function Codes

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

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

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

Virtual I/O functions, on the other hand, 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 also must be executed in a prescribed order. The normal order is to create and access a file, write information to that file, and then deaccess the file. Subsequently, when you access the file, you read the information, and then deaccess the file. You 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. (Note that the disk driver is responsible for any memory management functions of this type.) The size of the transfer does not affect the applicable quotas (direct I/O count, buffered I/O count, and AST count limit). These quotas refer to the number of outstanding I/O operations of each type, not the size of the I/O operation being performed.

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

Table 3–3 lists the logical, virtual, and physical disk I/O functions and their function codes. Section 1 describes the QIO level interface to the disk device ACP.

Table 3–3 Disk I/O Functions

Function Code and Arguments	Type[1]	Function Modifiers	Function
IO$_CREATE P1,[P2],- [P3],[P4],[P5]	V	IO$M_CREATE IO$M_ACCESS IO$M_DELETE	Create a directory entry or a file.
IO$_ACCESS P1, [P2],- [P3],[P4],[P5]	V	IO$M_CREATE IO$M_ACCESS	Search a directory for a specified file and access the file if found.
IO$_DEACCESS P1,[P2],- [P3],[P4],[P5]	V		Deaccess a file and if specified, write final attributes in the file header.
IO$_MODIFY P1,[P2], - [P3],[P4],[P5]	V		Modify the file attributes and /or allocation.
IO$_DELETE P1,[P2],- [P3],[P4],[P5]	V	IO$M_DELETE	Remove a directory entry and/or file header.
IO$_ACPCONTROL P1,- [P2],[P3],[P4],[P5]	V	IO$M_DMOUNT	Perform miscellaneous control functions.
IO$_READVBLK P1,P2,P3	V	IO$M_DATACHECK[2] IO$M_INHRETRY	Read virtual block.
IO$_READLBLK P1,P2,P3	L	IO$M_DATACHECK[2] IO$M_INHRETRY	Read logical block.
IO$_READPBLK P1,P2,P3	P	IO$M_DATACHECK[2] IO$M_INHRETRY IO$M_INHSEEK[3]	Read physical block.[5]
IO$_WRITEVBLK P1,P2,P3	V	IO$M_DATACHECK[2] IO$M_ERASE IO$M_INHRETRY	Write virtual block.
IO$_WRITELBLK P1,P2,P3	L	IO$M_DATACHECK[2] IO$M_ERASE IO$M_INHRETRY	Write logical 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 disks

Table 3–3 (Cont.) Disk I/O Functions

Function Code and Arguments	Type[1]	Function Modifiers	Function
IO$_WRITEPBLK P1,P2,P3	P	IO$M_DATACHECK[2] IO$M_ERASE IO$M_INHRETRY IO$M_INHSEEK[3] IO$M_DELDATA[4]	Write physical block.[5]
IO$_WRITECHECK P1,P2,P3	P		Verify data written to disk by a previous write QIO.[2]
IO$_SENSEMODE	L		Sense the device-dependent characteristics and return them in the I/O status block.
IO$_SENSECHAR	P		Sense the device-dependent characteristics and return them in the I/O status block.
IO$_FORMAT P1	P		Set density (RX02 only).
IO$_SEARCH P1	P		Search for specified block or sector (only for TU58).
IO$_PACKACK	P		Update UCB fields if RX02; initialize volume valid on other devices. Bring DSA units online.
IO$_AVAILABLE	P		Clear volume valid; make DSA units available.

---

[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 disks

Table 3-3 (Cont.) Disk I/O Functions

Function Code and Arguments	Type[1]	Function Modifiers	Function
IO$_UNLOAD	P		Clear volume valid; make DSA units available and spin down the volume.
IO$_SEEK P1	P		Seek to specified cylinder.[5]

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

[5] Not for DSA disks

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

- P1—the address of the file information block (FIB) descriptor.
- P2—the address of the file name string descriptor (optional). If specified, the name is entered in the directory specified by the FIB.
- P3—the address of the word that is to receive the length of the resulting file name string (optional).
- 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 Section 1 for more information on these functions.

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

- P1—the starting virtual address of the buffer that is to receive the data from a read operation; or, in the case of a write operation, the virtual address of the buffer that is to be written on the disk.
- P2—the number of bytes that are to be read from the disk, or written from memory to the disk. An even number must be specified if the controller is an RK611, RL11, RX211, or UDA50.
- P3—the starting virtual/logical disk address of the data to be transferred in a read operation; or, in a write operation, the disk address of the area that is to receive the data.

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

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

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

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

Note: DSA disk controllers provide controlled, assisted erasing for the IO$M_ERASE modifier (with virtual and logical write functions) but 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.
- 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, 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. The address is expressed as sector, track, and cylinder in the format shown in Figure 3–2. (On the RX02, the high word specifies the track number rather than the cylinder number.) Users should 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, the user is cautioned not to 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 will be positioned. The address is expressed as sector, track, and cylinder in the format shown in Figure 3–2.

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

- P1—the physical cylinder number where the disk head(s) will be positioned. The address is expressed in the format shown in Figure 3–3.

Figure 3–2 Starting Physical Address

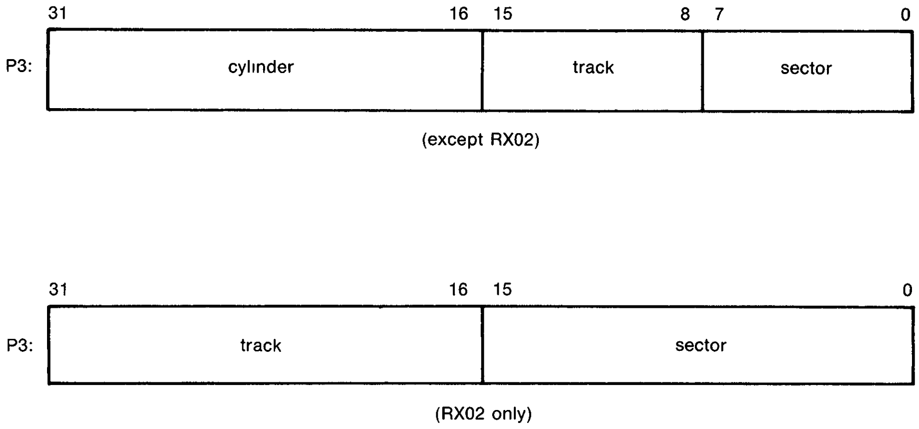
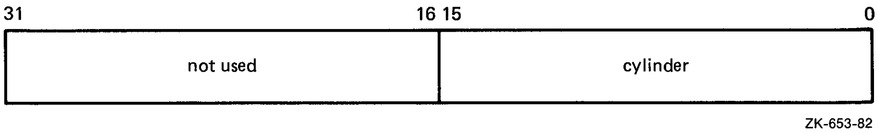

31
16
15
8
7
0
P3:
cylinder
track
sector
(except RX02)
31
16
15
0
P3:
track
sector
(RX02 only)
ZK-652-82

Figure 3–3 Physical Cylinder Number Format


31
16 15
0
not used
cylinder
ZK-653-82

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

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

3.4.1 Read

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

The VAX/VMS system provides three read function codes:

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

If a read virtual block function is 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 a file-structured device.

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

The data check function modifier (IO$M_DATACHECK) can be used with all read functions. If this modifier is specified, a data check operation is performed after the read operation completes. A data check operation is also performed if the volume that has been read, or the volume on which the file resides (virtual read), has the characteristic "data check all reads." Furthermore, a data check is performed after a virtual read if the file has the attribute "data check on read." The RX01 and RX02 drivers do not support the data check function.

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

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

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

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.

3.4.2 Write

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

The VAX/VMS system provides three write function codes:

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

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

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

The data check function modifier (IO$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. 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$_WRITEPBLK function is the same; write data is transferred to the disk. When a successful read is performed on this data, the status code SS$_RDDELDATA is returned in the I/O status block rather than the usual SS$_NORMAL status code.

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

3.4.3 Sense Mode

Sense mode operations obtain current disk device-dependent characteristics that are returned to the caller in the second longword of the I/O status block (see Figure 3–5). The VAX/VMS system provides two 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/function-dependent arguments are used with either function.

3.4.4 Set Density

The set density function assigns a new density to an entire RX02 floppy diskette. The diskette is also reformatted: new data address marks are written (single or double density) and all data fields are zeroed. Set density is a physical I/O function and requires the access privilege necessary to perform physical I/O. A single 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 may contain illegal data address marks in both densities. The diskette must then be completely reformatted and the function reissued.

3.4.5 Search

The search function positions a TU58 magnetic tape to the block specified. Search is a physical I/O function and requires the access privilege necessary to perform physical I/O. The VAX/VMS 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 was near the end of the tape and the next operation will be near the beginning of the tape, then the search operation can begin after the last operation completes, and the tape will rewind while the process is otherwise occupied. (Note that the search QIO is not completed until the search is completed. Consequently, if a $QIOW system service request is issued, the process will be held up until the search is completed.)

3.4.6 Pack Acknowledge

The pack acknowledge function sets the volume valid bit for all disk devices. Pack acknowledge is a physical I/O function and requires the access privilege to perform physical I/O. If directed to an RX02 drive, pack acknowledge also senses the floppy 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. A single function code is provided:

- IO$_PACKACK

This function code takes no function-dependent arguments.

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

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

3.4.7 Unload

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

- IO$_UNLOAD

This function takes no function-dependent arguments.

3.4.8 Available

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

- IO$_AVAILABLE

This function takes no function-dependent arguments.

3.4.9 Seek

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

3.4.10 Write Check

The write check function verifies that data was written to disk correctly. The data to be checked is addressed using physical disk addressing, that is, sector, track, and cylinder (see Figure 3–2). (If the request is directed to a DSA disk, the user must specify a logical block number, even though IO$_WRITECHECK is a physical I/O function.) The VAX/VMS system provides a single function code:

- IO$_WRITECHECK

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

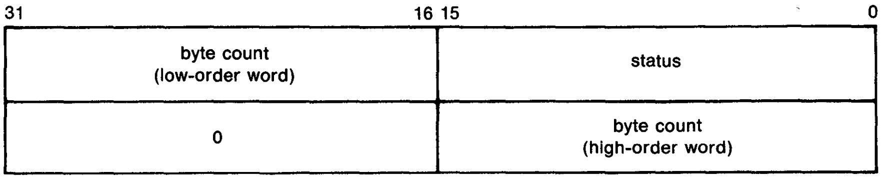
IO$_WRITECHECK is similar to the IO$M_DATACHECK function modifier for write QIOs, except that IO$_WRITECHECK does not write data to disk; it is specified after data is written by a separate write QIO. Nonprivileged processes may use the IO$M_DATACHECK modifier with IO$_WRITEVBLK (which does not require 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.

3.5 I/O Status Block

Figure 3–4 shows the I/O status block (IOSB) for all disk device QIO functions except Sense Mode. Figure 3–5 shows the I/O status block for the Sense Mode function. Appendix A lists the status returns for all functions and devices. (The *VAX/VMS System Messages and Recovery Procedures Reference Manual* provides explanations and suggested user actions for these returns.)

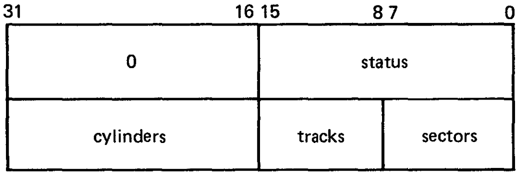
Figure 3–4 IOSB Contents



31 16	15 0
byte count (low-order word)	status
0	byte count (high-order word)

ZK-656-82

Figure 3–5 IOSB Contents - Sense Mode



31 16	15 8	7 0
0	status	
cylinders	tracks	sectors

ZK-657-82

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

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

3.6 Programming Example

This sample program (Example 3–1) provides an example of optimizing access time to a disk file. The program creates a file using VAX 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 3–1 Disk Program Example

```
; ********************************************************************
;
        .TITLE  Disk Driver Programming Example
        .IDENT  /01/

;
; Define necessary symbols.
;
        $FIBDEF                 ;Define file information block Offsets
        $IODEF                  ;Define I/O function codes
        $RMSDEF                 ;Define RMS-32 Return Status Values

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

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

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

;
; Allocate file information block descriptor.
;
FIB_DESCR:                                      ;
        .LONG   FIB$K_LENGTH                    ;Length of the file
                                                ;information block
        .LONG   FIB_BLOCK                       ;Address of the file
                                                ;information block
```

(Continued on next page)

Example 3–1 (Cont.) Disk Program Example

```
;
; Allocate File Name Block
;
;       A file name block is required by RMS-32 to return information
;       concerning a file (for example, the resultant file name string
;       after logical name translation and defaults have been applied).
;
NAM_BLOCK:                              ;
        $NAM                            ;

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

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

;
; Allocate space to store channel number returned by the $ASSIGN
; Channel system service.
;
DEVICE_CHANNEL:                         ;
        .BLKW   1                       ;

;
; Allocate device name string and descriptor.
;
DEVICE_DESCR:                           ;
        .LONG   20$-10$                 ;Length of device name string
        .LONG   10$                     ;Address of device name string
10$:    .ASCII  /SYS$DISK/              ;Device on which created file
                                        ;will reside
20$:                                    ;Reference label to calculate
                                        ;length

;
; Allocate file name string and define string length symbol.
;
FILE_NAME:                              ;
        .ASCII  /SYS$DISK:MYDATAFIL.DAT/        ;File name string
FILE_SIZE=.-FILE_NAME                   ;File name string length

;
; Allocate I/O status quadword storage.
;
IO_STATUS:                              ;
        .BLKQ   1                       ;

;
; Allocate output record buffer.
;
RECORD_BUFFER:                          ;
        .BLKB   512                     ;Record buffer is 512 bytes
```

(Continued on next page)

Example 3–1 (Cont.) Disk Program Example

```
;
; *********************************************************************
;
;                       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.
;
        $PUT    RAB = RAB_BLOCK         ;Put record in file
        BLBC    R0,30$                  ;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
```

(Continued on next page)

Example 3–1 (Cont.) Disk Program Example

```
;
; 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
        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
```

(Continued on next page)

Example 3–1 (Cont.) Disk Program Example

```
;
; 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.
;
        $QIOW_S CHAN = DEVICE_CHANNEL,- ;Write even-numbered record in
                -                       ;odd slot
                FUNC = #IO$_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$                     ;
```

(Continued on next page)

Example 3–1 (Cont.) Disk Program Example

```
;
; Check I/O completion status.
;
50$:    BLBC    R0,70$                  ;If low bit = 0, service
                                        ;failure
        MOVZWL  IO_STATUS,R0            ;Get final I/O completion
                                        ;status
        BLBC    R0,70$                  ;If low bit = 0, I/O function
        RSB                             ;failure
;
; Record number mismatch in data.
;
60$:    MNEGL   #4,R0                   ;Set dummy error status value
;
; All records have been read, verified, and odd/even pairs inverted
;
70$:    PUSHL   R0                      ;Save final status
        $QIOW_S CHAN = DEVICE_CHANNEL,- ;Deaccess file
                FUNC = #IO$_DEACCESS    ;I/O function is deaccess file
        $DASSGN_S CHAN = DEVICE_CHANNEL ;Deassign file device channel
        POPL    R0                      ;Retrieve final status
        RET                             ;

        .END    DISK_EXAMPLE
```

4 Laboratory Peripheral Accelerator Driver

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

To understand this section, the reader should also have access to a copy of the *LPA11-K Laboratory Peripheral Accelerator User's Guide.*

4.1 Supported Device

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

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

Table 4–1 lists the useful minimum and maximum LPA11-K configurations supported by VAX/VMS.

4.1.1 LPA11-K Modes of Operation

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

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

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

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

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

4.1.2 Errors

The LPA11-K returns three classes of errors:

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

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

- Power failure
- Device timeout
- Fatal error

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

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

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

4.2 Supporting Software

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

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

- They permit multiple LPA11-K subsystems on a single UNIBUS adapter.
- They operate as an integral part of the VAX/VMS operating system.
- They can be loaded on an operating VAX/VMS system without relinking the executive.

- They handle I/O requests, function dispatching, UNIBUS adapter map allocation, interrupts, and error reporting for multiple LPA11-K subsystems.
- The LPA11-K functions as a multibuffered device. Up to eight buffer areas can be defined per request. Up to eight requests can be handled simultaneously. Buffer areas can be reused after the data they contain is processed.
- Because the LPA11-K chains buffer areas automatically, a start data transfer request can transfer an infinite and noninterrupted amount of data.
- Multiple ASTs are dynamically queued by the driver to indicate when a buffer has been filled (the data is available for processing) or emptied (the buffer is available for new data).

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

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

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

- They execute, as separate processes, images that issue I/O requests. These I/O requests initiate microcode image loading, start the LPA11-K subsystem, and automatically configure the peripheral devices on the LPA11-K internal I/O bus.
- They can be executed at the request of the user or an operator.
- They can be executed at the request of other processes.
- They can be executed automatically when the system is initialized and on power recovery.

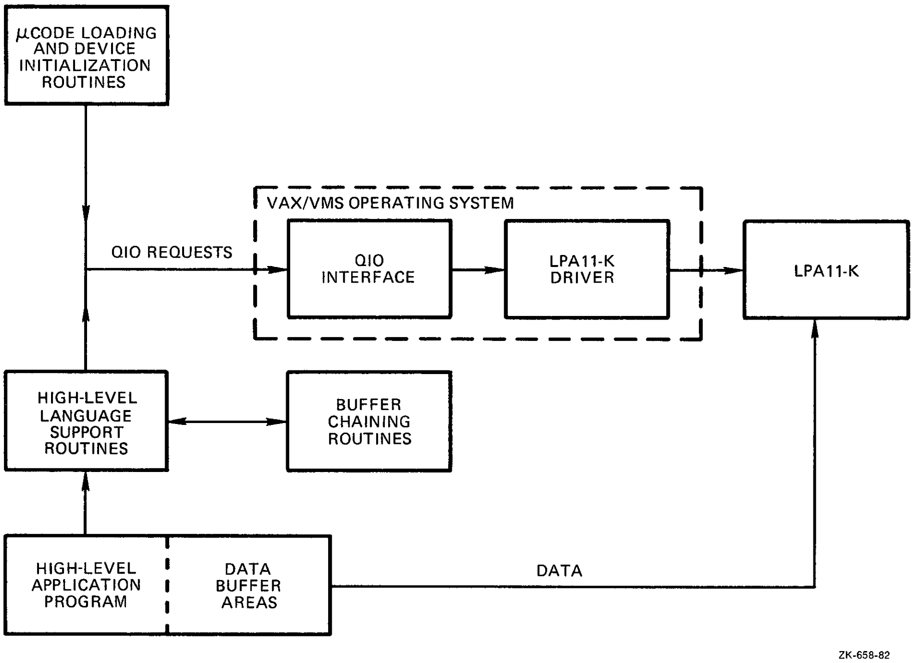
Figure 4–1 shows the relationship of the supporting software to the LPA11-K.

4.3 Device Information

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

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

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


μCODE LOADING AND DEVICE INITIALIZATION ROUTINES
VAX/VMS OPERATING SYSTEM
QIO REQUESTS
QIO INTERFACE
LPA11-K DRIVER
LPA11-K
HIGH-LEVEL LANGUAGE SUPPORT ROUTINES
BUFFER CHAINING ROUTINES
HIGH-LEVEL APPLICATION PROGRAM
DATA BUFFER AREAS
DATA
ZK-658-82

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

Table 4–2 LPA11-K Device-Independent Characteristics

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

[1] Defined by the $DEVDEF macro.

Table 4–3 LPA11-K Device-Dependent Characteristics

Field[1]	Meaning
LA$M_MCVALID LA$V_MCVALID	The load microcode I/O function (IO$_LOADMCODE) was performed successfully. LA$M_MCVALID is set by IO$_LOADMCODE. Each microword is verified by reading it back and comparing it with the specified value. LA$M_MCVALID is cleared if there is no match.
LA$V_MCTYPE LA$S_MCTYPE	The microcode type, set by the load microcode I/O function (IO$_LOADMCODE), is one of the following values:
	Value / **Meaning**
	LA$K_MRMCODE — Microcode type is in multirequest mode.
	LA$K_ADMCODE — Microcode type is in dedicated A/D mode.
	LA$K_DAMCODE — Microcode type is in dedicated D/A mode.
LA$V_CONFIG LA$S_CONFIG	The bit positions, set by the initialize I/O function (IO$_INITIALIZE), for the peripheral data acquisition devices on the LPA11-K internal I/O bus are one or more of the following:

Value	Meaning
LA$V_CLOCKA LA$M_CLOCKA	Clock A
LA$V_CLOCKB LA$M_CLOCKB	Clock B
LA$V_AD1 LA$M_AD1	A/D device 1
LA$V_AD2 LA$M_AD2	A/D device 2
LA$V_DA LA$M_DA	D/A device 1
LA$V_DIO1 LA$M_DIO1	Digital I/O buffer 1
LA$V_DIO2 LA$M_DIO2	Digital I/O buffer 2
LA$V_DIO3 LA$M_DIO3	Digital I/O buffer 3
LA$V_DIO4 LA$M_DIO4	Digital I/O buffer 4
LA$V_DIO5 LA$M_DIO5	Digital I/O buffer 5

---

[1] Defined by the $LADEF macro.

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

Field[1]	Meaning
LA$V_RATE LA$S_RATE	The Clock A rate, which is set by the set clock function (IO$_SETCLOCK), is one of the following values:

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

Field[1]	Meaning
LA$V_PRESET LA$S_PRESET	The Clock A preset value set by the set clock function (IO$_SETCLOCK). (The value is in two's complement form in the range 0 through 65,535.) The clock rate divided by the clock preset value yields the clock overflow rate.

[1] Defined by the $LADEF macro.

4.4 LPA11-K Function Codes

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

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

The first three functions are normally performed by the loader process, not by the user's data transfer program. See Section 4.7 for a description of the loader process interface.

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

4.4.1 Load Microcode

This I/O function resets the LPA11-K and loads an image of LPA11-K microcode. Physical I/O privilege is required. The VAX/VMS operating system defines a single function code:

- IO$_LOADMCODE—load microcode

The load microcode function takes three device/function-dependent arguments:

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

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

Each microword is verified by comparing it with the specified value in memory. If all words match, that is, if the microcode was loaded successfully, the driver sets the microcode valid bit (LA$V_MCVALID) in the device-dependent characteristics longword (see Table 4–3). If there is no match, SS$_DATACHECK is returned in the I/O status block and LA$V_MCVALID is cleared to indicate that the microcode was not properly loaded. If the microcode was loaded successfully, the driver stores one of the microcode type values (LAK_MRCODE, LAK_ADCODE, or LA$K_DAMCODE) in the characteristics longword.

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

4.4.2 Start Microprocessor

This I/O function resets the LPA11-K and starts (or restarts) the LPA11-K microprocessor. Physical I/O privilege is required. The VAX/VMS operating system defines a single function code:

- IO$_STARTMPROC—start microprocessor

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

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

SS$_CTRLERR SS$_DEVACTIVE SS$_MCNOTVALID

SS$_POWERFAIL SS$_TIMEOUT

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

4.4.3 Initialize LPA11-K

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

- IO$_INITIALIZE—initialize LPA11-K

The initialize LPA11-K function takes two device/function-dependent arguments:

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

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

The initialize function can return 10 error codes in the I/O status block (see Section 4.6):

SS$_BUFNOTALIGN
SS$_DEVCMDERR
SS$_IVMODE
SS$_TIMEOUT
SS$_CANCEL
SS$_INCLENGTH
SS$_MCNOTVALID
SS$_CTRLERR
SS$_INSFMAPREG
SS$_POWERFAIL

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

4.4.4 Set Clock

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

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

The VAX/VMS operating system defines a single function code:

- IO$_SETCLOCK—set clock

The set clock function takes three device/function-dependent arguments:

- P2—mode of operation. VAX/VMS defines the following clock start mode word (hexadecimal) values:

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

- P3—clock control and status. VAX/VMS defines the following clock status word (hexadecimal) values:

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

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

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

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

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

SS$_CANCEL
SS$_CTRLERR
SS$_DEVCMDERR
SS$_MCNOTVALID
SS$_POWERFAIL
SS$_TIMEOUT

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

4.4.5 Start Data Transfer Request

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

- IO$_STARTDATA—start data transfer request

The start data transfer request function takes one function modifier:

- IO$M_SETEVF—set event flag

The start data transfer request function takes four device/function-dependent arguments:

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

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

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

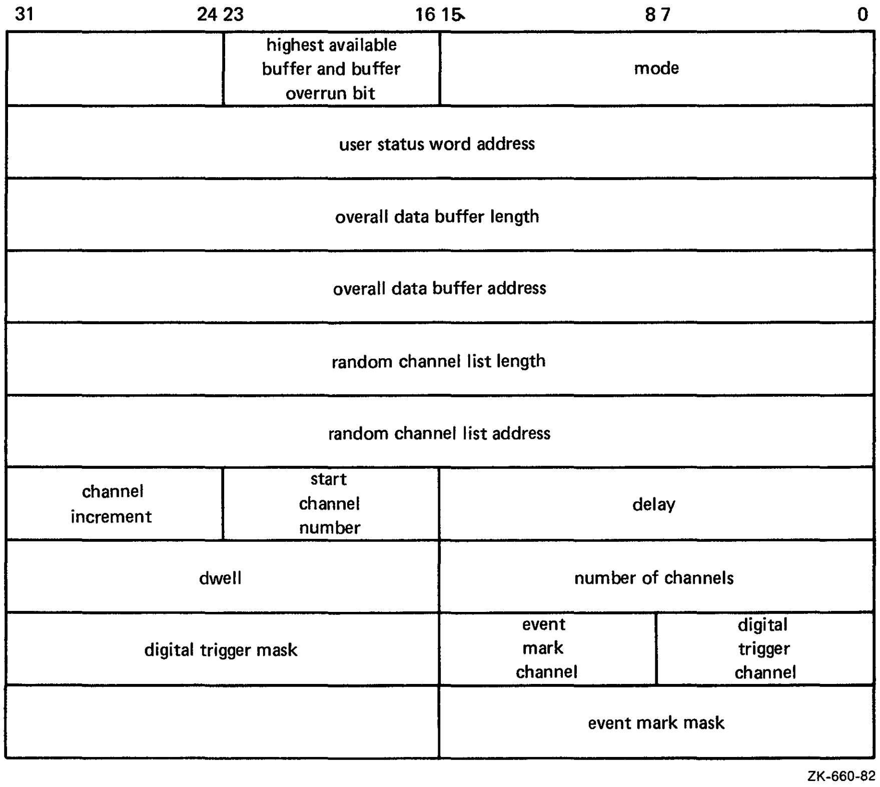
In the first longword of the data transfer command table, the first two bytes contain the LPA11-K start data transfer request mode word. (Section 3.4.1 of the *LPA11-K Laboratory Peripheral Accelerator User's Guide* describes the functions of this word.)

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

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

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

Figure 4–2 Data Transfer Command Table


31
24 23
16 15
8 7
0
highest available buffer and buffer overrun bit
mode
user status word address
overall data buffer length
overall data buffer address
random channel list length
random channel list address
channel increment
start channel number
delay
dwell
number of channels
digital trigger mask
event mark channel
digital trigger channel
event mark mask
ZK-660-82

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

The seventh through tenth longwords contain LPA11-K-specific sample parameters. The driver passes these parameters directly to the LPA11-K. (See Sections 3.4.7 through 3.4.12 of the *LPA11-K Laboratory Peripheral Accelerator User's Guide* for a detailed description of their functions.)

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

SS$_ABORT
SS$_BUFNOTALIGN
SS$_CANCEL
SS$_CTRLERR
SS$_DEVCMDERR
SS$_DEVREQERR
SS$_EXQUOTA
SS$_INCLENGTH
SS$_INSFBUFDP
SS$_INSFMAPREG
SS$_INSFMEM
SS$_MCNOTVALID
SS$_PARITY
SS$_POWERFAIL
SS$_TIMEOUT

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

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

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

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

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

4.4.6 LPA11-K Data Transfer Stop Command

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

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

4.5 High-Level Language Interface

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

4.5.1 High-Level Language Support Routines

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

Table 4–4 VAX Procedures for the LPA11-K

Class	Subroutine	Function
Sweep Control	LPA$ADSWP	Start A/D converter sweep
	LPA$DASWP	Start D/A converter sweep
	LPA$DISWP	Start digital input sweep
	LPA$DOSWP	Start digital output sweep
	LPA$LAMSKS	Specify LPA11-K controller and digital mask words
	LPA$SETADC	Specify channel select parameters
	LPA$SETIBF	Specify buffer parameters
	LPA$STPSWP	Stop sweep
Clock control	LPA$CLOCKA	Set Clock A rate
	LPA$CLOCKB	Set Clock B rate
	LPA$XRATE	Compute clock rate and preset value
Data Buffer Control	LPA$IBFSTS	Return buffer status
	LPA$IGTBUF	Return next available buffer
	LPA$INXTBF	Alter buffer order
	LPA$IWTBUF	Return next buffer or wait
	LPA$RLSBUF	Release buffer to LPA11-K
	LPA$RMVBUF	Remove buffer from device queue
Miscellaneous	LPA$CVADF	Convert A/D input to floating point
	LPA$FLT16	Convert unsigned integer to floating point
	LPA$LOADMC	Load microcode and initialize LPA11-K

4.5.1.1 Buffer Queue Control

This section is provided for informational purposes only. Normally, the user does not need to be concerned with the details of buffer queues.

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

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

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

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

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

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

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

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

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

4.5.1.2 Subroutine Argument Usage

Table 4–5 describes the general use of the subroutine arguments. The subroutine descriptions in the following sections contain additional information on argument usage. The (IBUF), (BUF), and (ICHN) (random channel list address) arguments must be aligned on specific boundaries. (The *VAX FORTRAN User's Guide* describes the alignment of FORTRAN arguments.)

Figure 4–3 Buffer Queue Control

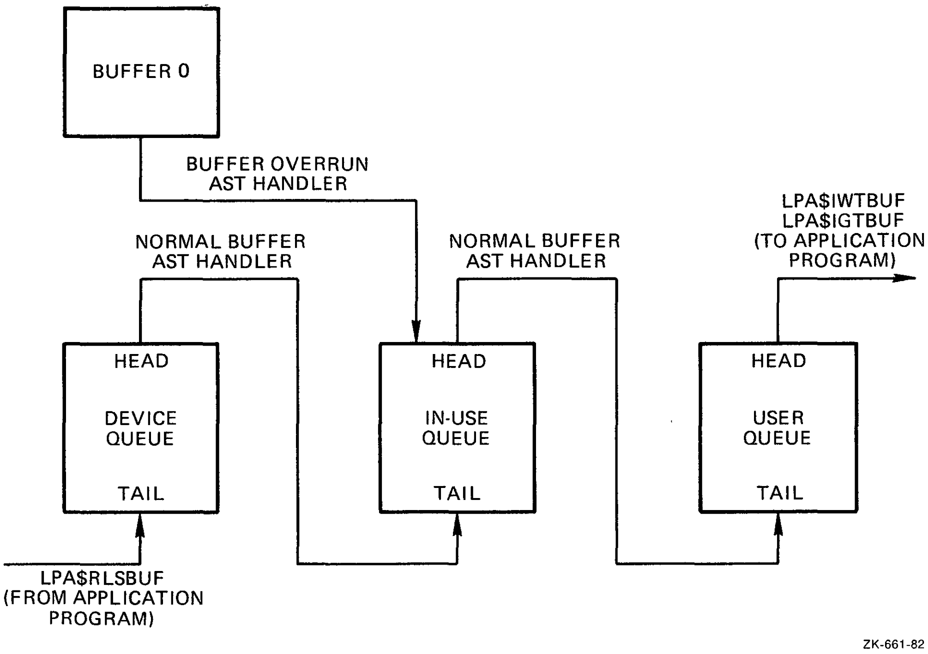

BUFFER 0
BUFFER OVERRUN AST HANDLER
NORMAL BUFFER AST HANDLER
NORMAL BUFFER AST HANDLER
LPA$IWTBUF
LPA$IGTBUF
(TO APPLICATION PROGRAM)
HEAD
DEVICE QUEUE
TAIL
HEAD
IN-USE QUEUE
TAIL
HEAD
USER QUEUE
TAIL
LPA$RLSBUF
(FROM APPLICATION PROGRAM)
ZK-661-82

Table 4–5 Subroutine Argument Usage

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

Table 4–5 (Cont.) Subroutine Argument Usage

Argument	Meaning
NBUF	Specifies the number of times the buffers are to be filled during the life of the request. If 0 (default) is specified, sampling is indefinite and must be stopped with the LPA$STPSWP subroutine. The NBUF argument length is one longword.
MODE	Specifies sampling options. MODE bit values are listed in the appropriate subroutine descriptions. The default is 0. MODE values can be added to specify several options. No options are mutually exclusive although not all bits may be applicable at the same time. The MODE argument length is one word.
IRATE	Specifies the clock rate:

Value	Meaning
−1	Direct-coupled Schmidt trigger 1 (Clock A only)
0	Clock B overflow or no rate
1	1 MHz
2	100 kHz
3	10 kHz
4	1 kHz
5	100 Hz
6	Schmidt trigger
7	Line frequency

Argument	Meaning
	The IRATE argument length is one longword.
IPRSET	Specifies the hardware clock preset value. This value is the two's complement of the desired number of clock ticks between clock interrupts. (The maximum value is 0, the two's complement of 65,536.) IPRSET can be computed by the LPA$XRATE subroutine. The IPRSET argument length is one word.
DWELL	Specifies the number of hardware clock overflows between sample sequences in multirequest mode. For example, if DWELL is 20 and NCHN is 3, then after 20 clock overflows one channel is sampled on each of the next three successive overflows; no sampling occurs for the next 20 clock overflows. This allows different users to use different sample rates with the same hardware clock overflow rate. In dedicated mode, the hardware clock overflow rate controls sampling and DWELL is not accessed. Default for DWELL is 1. The DWELL argument length is one word.

Table 4–5 (Cont.) Subroutine Argument Usage

Argument	Meaning

IEFN	Specifies the event flag number or completion routine address. The selected event flag is set at the end of each buffer transaction. If IEFN is 0 (default), event flag 22 is used.
	IEFN can also specify the address of a completion routine. This routine is called by the buffer management routine when a buffer is available and when the request is terminated, either successfully or with an error. The standard VAX/VMS calling and return sequences are used. The completion routine is called from an AST routine and is therefore at AST level.
	If IEFN specifies the address of a completion routine, the program must call the LPA$IGTBUF subroutine to obtain the next buffer. If IEFN specifies an event flag, the program must call the LPA$IWTBUF subroutine to obtain the next buffer and must use the %VAL operator:
	`,%VAL(3),        (Event flag 3)`
	`,BFRFULL,        (Address of completion routine)`
	The IEFN argument length is one longword.
	If multiple sweeps are initiated, they must use different event flags. The software does not enforce this policy.
	Event flag 23 is reserved for use by the LPA$CLOCKA and LPA$CLOCKB subroutines. If either of these subroutines is included in the user program, event flag 23 cannot be used. Also, if IEFN is defaulted, event flag 22 cannot be used in the user program.

LDELAY	Specifies the delay, in IRATE units, from the start event until the first sample is taken. The maximum value is 65,535; default is 1. The LDELAY argument length is one word. The LPA11-K supports the LDELAY argument in multirequest mode only.
ICHN	Specifies the number of the first I/O channel to be sampled. Default is channel 0. The ICHN argument length is one byte. The channel number is not the same as the channel assigned to the device by the $ASSIGN system service. The LPA11-K uses the channel number to specify the multiplexer address of an A/D, D/A, or digital I/O device on the LPA11-K internal I/O bus.
NCHN	Specifies the number of I/O device channels to sample in a sample sequence. Default is 1. If the NCHN argument is 1, the single channel bit is set in the mode word of the start request descriptor array (RDA) when the sweep is started. The RDA contains the information needed by the LPA11-K for each command (see the *LPA11-K Laboratory Peripheral Accelerator User's Guide*). The NCHN argument length is one word.
IND	Receives the VAX/VMS success or failure code of the call. The IND argument length is one longword.

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

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

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

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

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

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

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

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

IND	Returns the success or failure status: 0 = Error in call. Possible causes are: LPA$SETIBF subroutine was not previously called; LPA$RLSBUF subroutine was not previously called; size of data buffers disagrees with the size computed by the LPA$SETIBF subroutine call. 1 = successful sweep started nnn = VAX/VMS status code

4.5.3 LPA$DASWP — Initiate Synchronous D/A Sweep

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

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

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

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

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

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

IND	Returns the success or failure status: 0 = Error in call. Possible causes are: LPA$SETIBF subroutine was not previously called; LPA$RLSBUF subroutine was not previously called; size of data buffers disagrees with the size computed by the LPA$SETIBF subroutine call. 1 = successful sweep started nnn = VAX/VMS status code

4.5.4 LPA$DISWP — Initiate Synchronous Digital Input Sweep

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

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

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

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

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

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

IND Returns the success or failure status:

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

1 = successful sweep started

nnn = VAX/VMS status code

4.5.5 LPA$DOSWP — Initiate Synchronous Digital Output Sweep

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

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

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

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

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

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

IND Returns the success or failure status:

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

1 = successful sweep started

nnn = VAX/VMS status code

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

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

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

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

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

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

Argument descriptions are as follows:

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

IDSW	Specifies the digital START word mask. The IDSW argument length is one word.
IEMW	Specifies the event MARK word mask. The IEMW argument length is one word.
IND	Always equal to 1 (success). This argument is present for compatibility only.

4.5.7 LPA$SETADC — Set Channel Information for Sweeps

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

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

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

or

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

Argument descriptions are as follows:

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

4.5.8 LPA$SETIBF — Set IBUF Array for Sweeps

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

LPA$ADSWP
LPA$DASWP
LPA$DISWP
LPA$DOSWP
LPA$IBFSTS
LPA$IGTBUF
LPA$INXTBF
LPA$IWTBUF
LPA$RLSBUF

LPA$RMVBUF
LPA$SETADC
LPA$STPSWP

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

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

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

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

4.5.9 LPA$STPSWP — Stop In-Progress Sweep

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

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

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

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

IBUF	The IBUF array specified in the LPA$ADSWP, LPA$DASWP, LPA$DISWP, or LPA$DOSWP subroutine call that initiated the sweep.
IWHEN	Specifies when to stop the sweep. The VAX/VMS operating system defines the following values: 0 = Abort sweep immediately. Uses the $CANCEL system service. This is the default sweep stop. 1 = Stop sweep when the current buffer transaction is completed. (This is the preferred way to stop requests.)

IND	Receives a success or failure code in the standard VAX/VMS format: 1 = Success nnn = VAX/VMS error code issued by the $CANCEL system service

Note that when the LPA$STPSWP subroutine returns, the sweep may not yet be stopped. If it is necessary to wait until the sweep has stopped, the user can call the LPA$IWTBUF subroutine in a loop until it returns IBUFNO = –1 (see Section 4.5.16).

4.5.10 LPA$CLOCKA — Clock A Control

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

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

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

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

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

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

IPRSET	Specifies the clock preset value. Maximum of 16 bits. The LPA$XRATE subroutine can be used to calculate this value. The clock rate divided by the clock preset value yields the clock overflow rate.
IND	Receives a success or failure code as follows: 1 = Clock A set successfully nnn = VAX/VMS error code indicating an I/O error
NUM	Specifies the number to be appended to the logical name LPA11$. The default value is 0. This subroutine sets Clock A on the LPA11-K assigned to LPA11$num.

4.5.11 LPA$CLOCKB — Clock B Control

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

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

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

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

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

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

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

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

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

1 = Clock B operates in noninterrupt mode.

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

IND — Receives a success or failure code as follows:

1 = Clock B set successfully

nnn = VAX/VMS error code indicating an I/O error

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

4.5.12 LPA$XRATE — Compute Clock Rate and Preset Value

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

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

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

or

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

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

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

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

4.5.13 LPA$IBFSTS — Return Buffer Status

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

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

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

Argument descriptions are as follows:

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

4.5.14 LPA$IGTBUF — Return Buffer Number

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

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

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

or

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

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

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

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

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

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

4.5.15 LPA$INXTBF — Set Next Buffer To Use

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

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

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

or

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

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

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

4.5.16 LPA$IWTBUF — Return Next Buffer or Wait

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

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

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

or

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

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

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

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

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

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

4.5.17 LPA$RLSBUF — Release Data Buffer

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

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

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

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

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

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

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

4.5.18 LPA$RMVBUF — Remove Buffer from Device Queue

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

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

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

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

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

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

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

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

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

or

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

Argument descriptions are as follows:

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

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

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

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

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

or

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

Argument descriptions are as follows:

IVAL	An unsigned 16-bit integer.
VAL	Receives the converted value.

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

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

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

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

Argument descriptions are as follows:

ITYPE — The type of microcode to be loaded. The VAX/VMS operating system defines the following values:

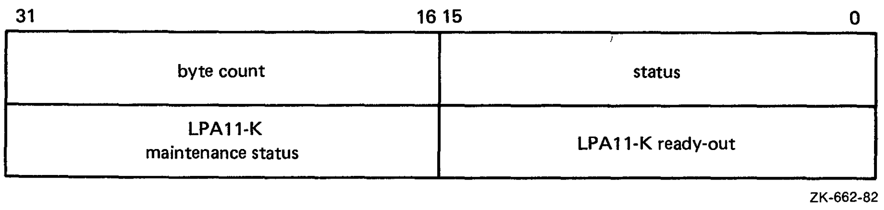
Value	Meaning
1	Multirequest mode; default value
2	Dedicated A/D mode
3	Dedicated D/A mode

NUM	The number to be appended to the logical name LPA11$. The default value is 0.
IND	Receives the completion status: 1 = Microcode loaded successfully nnn = VAX/VMS error code
IERROR	Provides additional error information. Receives the second longword of the I/O status block if SS$_CTRLERR, SS$_DEVCMDERR, or SS$_DEVREQERR is returned in IND. Otherwise, the contents of IERROR are undefined.

4.6 I/O Status Block

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

Figure 4–4 I/O Functions IOSB Content



VAX/VMS status values and the byte count are returned in the first longword. Status values are defined by the $SSDEF macro. The byte count is the number of bytes transferred by a IO$_LOADMCODE request. If SS$_CTRLERR, SS$_DEVCMDERR, or SS$_DEVREQERR is returned in the status word, the second longword contains the LPA11-K ready-out register and LPA11-K maintenance status register values present at the completion of the request. The high byte of the ready-out register contains the specific

LPA11-K error code (see Appendix A of the *LPA11-K Laboratory Peripheral Accelerator User's Guide*). Appendix A of this manual lists the status returns for LPA11-K I/O functions. (The *VAX/VMS System Messages and Recovery Procedures Reference Manual* provides explanations and suggested user actions for these returns.)

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

4.7 Loading LPA11-K Microcode

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

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

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

4.7.1 Microcode Loader Process

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

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

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

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

4.7.2 Operator Process

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

```
devname/type
```

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

- /MULTI_REQUEST—multirequest mode
- /ANALOG_DIGITAL—dedicated A/D mode
- /DIGITAL_ANALOG—dedicated D/A mode

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

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

4.8 RSX–11M/M–PLUS and VAX/VMS Differences

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

4.8.1 Alignment and Length

In VAX/VMS:

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

4.8.2 Status Returns

In VAX/VMS:

- The I/O status block (IOSB) length is eight bytes; numeric values of errors are different.
- Several routines return:

  1 = Success

  0 = Failure detected in support routine

  nnn = VAX/VMS status code; failure detected in system service

4.8.3 Sweep Routines

In VAX/VMS:

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

4.8.4 General

In VAX/VMS:

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

4.9 Programming Examples

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

The *Writing a Device Driver for VAX/VMS* volume contains information that is applicable to LPA11-K programming.

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

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

Table 4–8 Program A Variables

Variable	Description
BUFFER	The data buffer array. BUFFER is a common area to guarantee longword alignment.
IBUF	The LPA11-K high-level language procedures use the IBUF array for local storage.
BUFNUM	BUFNUM contains the buffer number returned by LPA$IWTBUF. In this example, the possible values are 0, 1, and 2.
ISTAT	ISTAT contains the status return from the high-level language calls.

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

```
C  ******************************************************************
C
C                              PROGRAM A
C
C  ******************************************************************
       INTEGER*2      BUFFER(1000,0:2),IOSB(4)
       INTEGER*4      IBUF(50),ISTAT,BUFNUM
       COMMON/AREA1/BUFFER
       EQUIVALENCE    (IOSB(1),IBUF(1))
C
C  SET CLOCK RATE TO 1 KHZ, CLOCK PRESET TO -10
C
       CALL LPA$CLOCKA(4,-10,ISTAT)
       IF (.NOT. ISTAT) GO TO 950
C
C  INITIALIZE IBUF ARRAY FOR SWEEP
C
       CALL LPA$SETIBF(IBUF,ISTAT,,BUFFER(1,0),BUFFER(1,1),BUFFER(1,2))
       IF (.NOT. ISTAT) GO TO 950
C
C  RELEASE ALL THE BUFFERS. NOTE USE OF BUFFER NUMBERS RATHER THAN
C  BUFFER NAMES.
C
       CALL LPA$RLSBUF(IBUF,ISTAT,0,1,2)
       IF (.NOT. ISTAT) GO TO 950
C
C  START A/D SWEEP
C
       CALL LPA$ADSWP(IBUF,1000,50,,,,,,ISTAT)
       IF (.NOT. ISTAT) GO TO 950
C
C  GET NEXT BUFFER FILLED WITH DATA. IF BUFNUM IS NEGATIVE, THERE
C  ARE NO MORE BUFFERS AND THE SWEEP IS STOPPED.
C
100    BUFNUM = LPA$IWTBUF(IBUF)
       IF (BUFNUM .LT. 0) GO TO 800
C
C  PROCESS DATA IN BUFFER(1,BUFNUM) TO BUFFER (1000,BUFNUM)
       .
       .
       .
(Application-dependent code is inserted at this point)
       .
       .
       .
C  RELEASE BUFFER TO BE FILLED AGAIN
C
200    CALL LPA$RLSBUF(IBUF,ISTAT,BUFNUM)
       IF (.NOT. ISTAT) GO TO 950
       GO TO 100
C
C  THERE ARE NO MORE BUFFERS TO PROCESS. CHECK TO ENSURE THAT THE
C  SWEEP ENDED SUCCESSFULLY. IOSB(1) CONTAINS EITHER 1 OR A
C  VAX/VMS STATUS CODE.
C
800    IF (.NOT. IOSB(1)) CALL LIB$STOP(%VAL(IOSB(1)))
       PRINT *,'SUCCESSFUL COMPLETION'
       GO TO 2000
```

(Continued on next page)

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

```
C
C  ERROR RETURN FROM SUBROUTINE. ISTAT CONTAINS EITHER 0 OR
C  VAX/VMS ERROR CODE.
C
950     IF (ISTAT .NE. 0) CALL LIB$STOP(%VAL(ISTAT))
        PRINT  *,'ERROR IN LPA11-K SUBROUTINE CALL'
2000    STOP
        END
C  ******************************************************************
```

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

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

- Program-requested loading of LPA11-K microcode
- Setting the clock at a specified rate
- Use of nondefault arguments whenever possible
- An A/D sweep that uses an event flag
- A D/A sweep that uses a completion routine
- Buffer overrun set (buffer overrun is a nonfatal error)
- Random channel list (RCL) addressing
- Sequential channel addressing

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

Table 4–9 Program B Variables

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

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

```
C  *******************************************************************
C
C                             PROGRAM B
C
C  *******************************************************************
       EXTERNAL FILLBF
       REAL*4 LPA$XRATE
       INTEGER*2 AD(500,0:7),DA(2000,0:1),RCL(5),MODE,IPRSET
       INTEGER*2 ADIOSB(4),DAIOSB(4)
       INTEGER*4 IBUFAD(50),IBUFDA(50),LAMSKB(2)
       INTEGER*4 ISTAT,IERROR,IRATE,BUFNUM
       REAL*4 PERIOD
       COMMON  /SWEEP/AD,DA,IBUFAD,IBUFDA
       EQUIVALENCE (IBUFAD(1),ADIOSB(1)),(IBUFDA(1),DAIOSB(1))
       PARAMETER MULTI=1, HBIT='8000'X, LSTCHN=HBIT+7
C
C  SET UP RANDOM CHANNEL LIST. NOTE THAT THE LAST WORD MUST HAVE BIT
C  15 SET.
C
       DATA RCL/2,6,3,4,LSTCHN/
C  *******************************************************************
C
C  LOAD MULTIREQUEST MODE MICROCODE AND SET THE CLOCK OVERFLOW RATE
C  TO 5 KHZ
C
C  *******************************************************************
C
C  LOAD MICROCODE ON LPA11-K ASSIGNED TO LPA11$3
C
       CALL LPA$LOADMC(MULTI,3,ISTAT,IERROR)
       IF (.NOT. ISTAT) GO TO 5000
C
C  COMPUTE CLOCK RATE AND PRESET. SET CLOCK 'A' ON LPA11-K
C  ASSIGNED TO LPA11$3.
C
       PERIOD = LPA$XRATE(.0002,IRATE,IPRSET,0)
       IF (PERIOD .EQ. 0.0) GO TO 5500
       CALL LPA$CLOCKA(IRATE,IPRSET,ISTAT,3)
       IF (.NOT. ISTAT) GO TO 5000
C  *******************************************************************
C
C  SET UP FOR A/D SWEEP
C
C  *******************************************************************
C
C  INITIALIZE IBUF ARRAY. NOTE THE USE OF THE LAMSKB ARGUMENT BECAUSE
C  THE LPA11-K ASSIGNED TO LPA11$3 IS USED.
C
       CALL LPA$SETIBF(IBUFAD,ISTAT,LAMSKB,AD(1,0),AD(1,1),AD(1,2),
       1               AD(1,3),AD(1,4),AD(1,5),AD(1,6),AD(1,7))
       IF (.NOT. ISTAT) GO TO 5000
       CALL LPA$LAMSKS(LAMSKB,3)
C
C  SET UP RANDOM CHANNEL LIST SAMPLING (20 SAMPLES IN A SAMPLE
C  SEQUENCE)
C
       CALL LPA$SETADC(IBUFAD,,RCL,20,0,ISTAT)
       IF (.NOT. ISTAT) GO TO 5000
```

(Continued on next page)

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

```
C
C  RELEASE BUFFERS FOR A/D SWEEP. NOTE THAT BUFFER 0 IS NOT
C  RELEASED BECAUSE BUFFER OVERRUN WILL BE SPECIFIED AS NONFATAL.
C
        CALL LPA$RLSBUF(IBUFAD,ISTAT,1,2,3,4,5,6,7)
        IF (.NOT. ISTAT) GO TO 5000
C  ********************************************************************
C
C  SET UP FOR D/A SWEEP
C
C  ********************************************************************
C
C  NOTE THAT THE SAME LAMSKB ARRAY CAN BE USED BECAUSE THE LAMSKB
C  CONTENTS APPLY TO BOTH A/D AND D/A SWEEPS
C
        CALL LPA$SETIBF(IBUFDA,ISTAT,LAMSKB,DA(1,0),DA(1,1))
        IF (.NOT. ISTAT) GO TO 5000
C
C  SET UP SAMPLING PARAMETERS AS FOLLOWS:  INITIAL CHANNEL = 1.
C  NUMBER OF CHANNELS SAMPLED EACH SAMPLE SEQUENCE = 2, CHANNEL
C  INCREMENT = 2, THAT IS, SAMPLE CHANNELS 1 AND 3 EACH SAMPLE
C  SEQUENCE.
C
        CALL LPA$SETADC(IBUFDA,,1,2,2,ISTAT)
        IF (.NOT. ISTAT) GO TO 5000
C
C  FILL BUFFERS WITH DATA FOR OUTPUT TO D/A
C
        .
        .
        .
(Application-dependent code is inserted here to fill buffers
DA(1,0) through DA(2000,0) and DA(1,1) through DA(2000,1) with data)
        .
        .
        .
C
C  RELEASE BUFFERS FOR D/A SWEEP
C
        CALL LPA$RLSBUF (IBUFDA,ISTAT,0,1)
        IF (.NOT. ISTAT) GO TO 5000
C  ******************************************************************
C
C  START BOTH SWEEPS
C
C  ******************************************************************
C
C  START A/D SWEEP. MODE BITS SPECIFY BUFFER OVERRUN IS NONFATAL AND
C  MULTIREQUEST MODE. SWEEP ARGUMENTS SPECIFY 500 SAMPLES/BUFFER,
C  INDEFINITE SAMPLING, DWELL = 10 CLOCK OVERFLOWS, SYNCHRONIZE USING
C  EVENT FLAG 15, AND A DELAY OF 50 CLOCK OVERFLOWS.
C
        MODE = 16384 + 64
        CALL LPA$ADSWP(IBUFAD,500,0,MODE,10,%VAL(15),50,,,ISTAT)
        IF (.NOT. ISTAT) GO TO 5000
C
C  START D/A SWEEP.  MODE SPECIFIES MULTIREQUEST MODE.  OTHER
C  ARGUMENTS SPECIFY 2000 SAMPLES/BUFFER, FILL 15 BUFFERS, DWELL = 25
C  CLOCK OVERFLOWS, SYNCHRONIZE BY CALLING THE COMPLETION ROUTINE
C  'FILLBF', AND DELAY = 10 CLOCK OVERFLOWS.  (SEE THE FILLBF LISTING
C  AFTER THE PROGRAM B LISTING.)
C
        MODE = 64
        CALL LPA$DASWP(IBUFDA,2000,15,MODE,25,FILLBF,10,,,ISTAT)
        (.NOT.  ISTAT) GO TO 5000
```

(Continued on next page)

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

```
C *******************************************************************
C
C WAIT FOR AN A/D BUFFER AND THEN PROCESS THE DATA IT CONTAINS.  D/A
C BUFFERS ARE FILLED ASYNCHRONOUSLY BY THE COMPLETION ROUTINE FILLBF.
C
C *******************************************************************
C
C WAIT FOR A BUFFER TO BE FILLED BY A/D.  IF BUFNUM IS LESS THAN
C ZERO, THE SWEEP HAS STOPPED (EITHER SUCCESSFULLY OR WITH AN ERROR).
C
100     BUFNUM = LPA$IWTBUF(IBUFAD)
        IF (BUFNUM .LT.  0) GO TO 1000
C
C THERE IS A/D DATA IN AD(1,BUFNUM) THROUGH AD(500,BUFNUM)
C
        .
        .
        .
(Process the A/D data with the application-dependent code inserted
here)
        .
        .
        .
C
C ASSUME SWEEP SHOULD BE STOPPED WHEN THE LAST SAMPLE IN BUFFER
C EQUALS 0.  NOTE THAT THE SWEEP ACTUALLY STOPS WHEN THE BUFFER
C CURRENTLY BEING FILLED IS FULL.  ALSO NOTE THAT LPA$IWTBUF
C CONTINUES TO BE CALLED UNTIL THERE ARE NO MORE BUFFERS TO PROCESS.
C
        IF (AD(500,BUFNUM) .NE.  0) GO TO 200
        CALL LPA$STPSWP(IBUFAD,1,ISTAT)
        IF (.NOT.  ISTAT) GO TO 5000
C
C AFTER THE DATA HAS BEEN PROCESSED, THE BUFFER IS RELEASED TO BE
C FILLED AGAIN.  THEN THE NEXT BUFFER IS OBTAINED FROM A/D.
C
200     CALL LPA$RLSBUF(IBUFAD,ISTAT,BUFNUM)
        IF (.NOT.  ISTAT) GO TO 5000
        GO TO 100
C
C ENTER HERE WHEN A/D SWEEP HAS ENDED.  CHECK FOR ERROR OR
C SUCCESSFUL END.  (NOTE: ASSUME THAT THE D/A SWEEP HAS ALREADY
C ENDED - SEE COMPLETION ROUTINE FILLBF)
C
1000    IF(ADIOSB(1)) GO TO 6000
        CALL LIB$STOP(%VAL(ADIOSB(1)))
C
C ENTER HERE IF THERE WAS AN ERROR RETURNED FROM ONE OF THE
C LPA11-K HIGH-LEVEL LANGUAGE CALLS.  ISTAT CONTAINS EITHER 0
C OR A VAX/VMS STATUS CODE.
C
5000    IF (ISTAT .NE.  0) CALL LIB$STOP (%VAL(ISTAT))
5500    PRINT *,'ERROR IN LPA11-K SUBROUTINE CALL'
        GO TO 7000
6000    PRINT *,'SUCCESSFUL COMPLETION'
7000    STOP
        END
```

(Continued on next page)

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

```
C  ******************************************************************
C
C  SUBROUTINE FILLBF
C
C  ******************************************************************
C
C  THE FILLBF SUBROUTINE IS CALLED WHENEVER THE D/A HAS EMPTIED A
C  BUFFER, AND THAT BUFFER IS AVAILABLE TO BE REFILLED.  THIS
C  SUBROUTINE GETS THE BUFFER, FILLS IT, AND RELEASES IT BACK TO THE
C  LPA11-K.  NOTE THAT THE D/A SWEEP IS STOPPED AUTOMATICALLY AFTER
C  15 BUFFERS HAVE BEEN FILLED.  ALSO NOTE THAT FILLBF IS CALLED BY
C  AN AST HANDLER.  IT IS THEREFORE CALLED ASYNCHRONOUSLY FROM THE
C  MAIN PROGRAM AT AST LEVEL.  CARE SHOULD BE EXERCISED WHEN ACCESSING
C  VARIABLES THAT ARE COMMON TO BOTH LEVELS.
C
        INTEGER*2  AD(500,0:7),DA(2000,0:1),DAIOSB(4)
        INTEGER*4  IBUFAD(50),IBUFDA(50),BUFNUM,ISTAT
        EQUIVALENCE  (IBUFDA(1),DAIOSB(1))
        COMMON  /SWEEP/AD,DA,IBUFAD,IBUFDA
C
C  GET BUFFER NUMBER OF NEXT BUFFER TO FILL
C
        BUFNUM = LPA$IGTBUF(IBUFDA)
        IF (BUFNUM .LT.  0) GO TO 3000
C
C  FILL BUFFER WITH DATA FOR OUTPUT TO D/A
        .
        .
        .
(Application-dependent code is inserted here to fill buffer
DA(1,BUFNUM) through DA(2000,BUFNUM) with data)
        .
        .
        .
C
C  RELEASE BUFFER
C
        CALL LPA$RLSBUF(IBUFDA,ISTAT,BUFNUM)
        GO TO 4000
C
C  CHECK FOR SUCCESSFUL END OF SWEEP
C
3000    IF(DAIOSB(1)) GO TO 4000
C
C  ERROR IN SWEEP
C
        CALL LIB$STOP(%VAL(DAIOSB(1)))
4000    RETURN
        END
C  ******************************************************************
```

4.9.3 LPA11-K QIO Functions Program (Program C)

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

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

```
; *******************************************************************
;
;                              PROGRAM C
;
; *******************************************************************
        .TITLE  LPA11-K EXAMPLE PROGRAM
        .IDENT  /V01/
        .PSECT  LADATA,LONG
IOSB:   .BLKQ  1                         ; I/O STATUS BLOCK
COUNT:  .LONG  0                         ; COUNT OF BUFFERS FILLED
CBUFF:                                   ; COMMAND BUFFER FOR START
                                         ; DATA QIO
        .WORD  ^X20A                     ; MODE = SEQUENTIAL CHANNEL
                                         ; ADDRESSING, A/D, MULTI-
                                         ; REQUEST MODE
        .WORD  3                         ; VALID BUFFER MASK (4
                                         ; BUFFERS)
        .LONG  USW                       ; USER STATUS WORD ADDRESS
        .LONG  4000                      ; AGGREGATE BUFFER LENGTH
        .LONG  DATA_BUFFER0              ; ADDRESS OF DATA BUFFERS
        .LONG  0                         ; NO RANDOM CHANNEL LIST
                                         ; LENGTH
        .LONG  0                         ; NO RANDOM CHANNEL LIST
                                         ; ADDRESS
        .WORD  10                        ; DELAY
        .BYTE  0                         ; START CHANNEL
        .BYTE  1                         ; CHANNEL INCREMENT
        .WORD  16                        ; NUMBER OF SAMPLES IN
                                         ; SAMPLE SEQUENCE
        .WORD  1                         ; DWELL
        .BYTE  0                         ; START WORD NUMBER
        .BYTE  0                         ; EVENT MARK WORD
        .WORD  0                         ; START WORD MASK
        .WORD  0                         ; EVENT MARK MASK
        .WORD  0                         ; FILLS OUT COMMAND BUFFER
USW:    .WORD  0                         ; USER STATUS WORD
        .ALIGN LONG                      ; BUFFERS MUST BE
                                         ; LONGWORD ALIGNED
DATA_BUFFER0: .BLKW    500               ; DATA BUFFERS
DATA_BUFFER1: .BLKW    500
DATA_BUFFER2: .BLKW    500
DATA_BUFFER3: .BLKW    500
DEVNAME: .ASCID /LAA0/
CHANNEL: .BLKW  1                        ; CONTAINS CHANNEL NUMBER
        .PSECT  LACODE,NOWRT
START:  .ENTRY  START,^m<>
        $ASSIGN_S DEVNAM=DEVNAME,CHAN=CHANNEL ; ASSIGN CHANNEL
        BLBS    R0,5$                    ; NO ERROR
        BRW     ERROR                    ; ERROR
5$:                                      ; SET CLOCK OVERFLOW RATE
                                         ; TO 2 KHZ.  (1 MHZ RATE
                                         ; DIVIDED BY 500 PRESET)
        $QIOW_S ,CHAN=CHANNEL,FUNC=#IO$_SETCLOCK,-
                IOSB=IOSB,,,,P2=#1,P3=#^X143,P4#-500
        BLBC    R0,ERROR                 ; ERROR
        MOVZWL  IOSB,R0                  ; PICK UP I/O STATUS
```

(Continued on next page)

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

```
        BLBC    R0,ERROR                ; ERROR
                                        ; START DATA TRANSFER
        CLRW    USW                     ; CLEAR USW (START WITH
                                        ; BUFFER 0)
        MOVL    #100,COUNT              ; FILL 100 BUFFERS
        $QIOW_S ,CHANNEL,#IO$_STARTDATA,-
                IOSB=IOSB,,,P1=CBUFF,P2=#40,P3=#BFRAST
        BLBC    R0,ERROR                ; ERROR
; NOTE THAT THE QIO WAITS UNTIL IT FINISHES.  NORMALLY, THE DATA IS
; PROCESSED HERE AS THE BUFFERS ARE FILLED.  CHECK FOR ERROR WHEN
; THE QIO COMPLETES.
        MOVZWL   IOSB,R0                ; PICK UP I/O STATUS
        BLBC     R0,ERROR               ; ERROR
        RET                             ; ALL DONE - EXIT
ERROR:                                  ; ENTER HERE IF ERROR.
                                        ; STATUS IN R0.
        PUSHL    R0                     ; PUSH ONTO STACK
        CALLS    #1,G^LIB$STOP          ; SIGNAL ERROR
BFRAST: BFRAST,m^<>                     ; BUFFER AST ROUTINE.
                                        ; BFRAST IS CALLED WHENEVER
                                        ; A BUFFER IS FILLED.
        .WORD    0
        INCB     USW+1                  ; ADD 1 TO BUFFER NUMBER
        CMPZV    #0,#3,USW+1,#3         ; HANDLE WRAPAROUND
        BLEQ     10$
        CLRB     USW+1                  ; USE BUFFER 0
10$:    DECL     COUNT                  ; DECREMENT BUFFER COUNT
        BGTR     20$
        BISB     #^X40,USW+1            ; ENOUGH BUFFERS FILLED -
                                        ; SET STOP BIT
20$:    BICB     #^X80,USW+1            ; CLEAR DONE BIT
        RET
        .END     START
; ********************************************************************
```

5 Line Printer Driver

This section describes the use of the line printer drivers LPDRIVER and LCDRIVER.

5.1 Supported Line Printer Devices

The following sections describe the line printer controllers and line printers supported by the VAX/VMS operating system.

5.1.1 LP11 Line Printer Controller

The LP11 line printer controller provides an interface between the VAX UNIBUS adapter and the line printer. The LP11 performs the following functions:

- Synchronizes single-character data transfers from the UNIBUS to the printer
- Informs the VAX/VMS operating system about printer status
- Enables the printer to gain control of the UNIBUS to report interrupts

5.1.2 DMF32 and DMB32 Line Printer Controllers

The DMF32 and DMB32 line printer controllers provide a direct memory access (DMA) interface between the VAX UNIBUS adapter (for the DMF32), or the VAXBI adapter (for the DMB32), and the line printer. The DMF32 /DMB32 optionally perform the following functions:

- Tab expansion
- Carriage control
- Line wrapping and truncation
- Case conversion
- Passall mode
- Printall mode

5.1.3 LP27 Line Printer

The LP27 line printer is a high-speed, 132-column line printer available in several models with either a 64- or 96-character ASCII print set. The LP27-U is a fully buffered model that operates at a standard speed of up to 1200 lines per minute. Forms with up to six parts can be used for multiple copies. A version of the LP27 is available for operation of the printer up to 24.5 meters (1000 feet) from the host.

5.1.4 LA11 DECprinter I

The LA11 DECprinter I is a medium-speed printer that operates at a standard speed of 180 characters per second. It incorporates such features as a forms length switch to set the top of form to any of 11 common lengths, a paper-out switch and alarm, and a variable forms width. The LA11 uses a 96-character ASCII set; the column width is 132 characters.

5.1.5 LN01 Laser Page Printer

The LN01 laser page printer is a nonimpact printer that employs laser technology to produce high-quality print. Using electrophotographic imaging and xerographic printing, the LN01 prints one page at a time at a rate of 12 pages a minute. The print resolution of 300 x 300 dots per square inch produces perfectly formed characters of even density and alignment. The LN01 uses two 188-character fixed-space fonts; the column width is 132 characters.

5.1.6 LN03 Laser Page Printer

The LN03 laser page printer is a table-top, nonimpact page printer that uses laser imaging and xerographic printing techniques. The LN03 has a printing speed of 8 pages per minute with a print resolution of 300 x 300 dots per square inch. Four built-in fonts are available. Several column widths, including 80 or 132 characters, are also available.

5.2 Driver Features and Capabilities

The line printer drivers provide output character formatting and error recovery. These features are described in the following sections.

5.2.1 Output Character Formatting

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

- Rubouts are discarded.
- Tabs move the horizontal print position to the next MODULO (8) position unless the LP$M_TAB characteristic is clear.
- All lowercase alphabetic characters are converted to uppercase before printing (unless the characteristic specifying lowercase characters is set; see Section 5.4.3 and Table 5–2).
- On printers where the line feed, form feed, vertical tab, and return characters empty the printer buffer, returns are held back and output only if the next character is not a form feed, line feed, or vertical tab. Returns are always output on units that have the LP$M_CR characteristic set (see Section 5.4.3 and Table 5–2).

- The horizontal print position is incremented on the output of all characters, including the space character. Characters are discarded if the horizontal print position is equal to or greater than the carriage width, unless the LP$M_WRAP characteristic is set or the LP$M_TRUNCATE characteristic is clear (see Section 5.3).
- On printers without mechanical form feed (the form feed function characteristic is not set; see Section 5.4.3 and Table 5–2), a form feed is converted to multiple line feeds. The number of line feeds is based on the current line count and the page length.
- Print lines are counted and returned to the caller in the second longword of the I/O status block.

5.2.2 Error Recovery

The VAX/VMS line printer drivers perform the following error recovery operations:

- If the printer is off line for 30 seconds, a "device not ready" message is sent to the system operator process.
- If the printer runs out of paper or has a fault condition, a "device not ready" message is sent to the system operator after 30 seconds. Successive messages, if they occur, are sent 1, 2, 4, 8,... minutes after the initial message.
- The current operation is retried every two seconds to test for a changed situation, for example, the printer coming on line.
- The current I/O operation can be canceled at the next timeout without the printer being on line.
- When the printer comes on line, device operation resumes automatically.

5.3 Device Information

The user process can obtain information on printer characteristics by using the Get Device/Volume Information ($GETDVI) system service. (See the *VAX/VMS System Services Reference Manual* in the *VAX/VMS System Routines Reference Volume*).

$GETDVI returns line printer characteristics when you specify the item codes DVI$_DEVCHAR and DVI$_DEVDEPEND. Tables 5–1 and 5–2 list these characteristics. The $DEVDEF macro defines the device-independent characteristics; the $LPDEF macro defines the device-dependent characteristics. DVI$_DEVDEPEND returns a longword field that contains the device-dependent characteristics in the three low-order bytes and the page length in the high-order byte. Maximum page length is 255.

DVI$_DEVTYPE and DVI$_DEVCLASS return the device type and class names, which are defined by the $DCDEF macro. The device type is a value that corresponds to the printer, for example, LP$_LP27 or LP$_LA11. The device class for printers is DC$_LP. DVI$_DEVBUFSIZ returns the page width, which is a value in the range of 0 through 255 on a DMF32 controller and 0 through 65535 on an LP11 or DMB32 controller.

Table 5–1 Printer Device-Independent Characteristics

Characteristic[1]	Meaning
Dynamic Bits (Conditionally Set)	
DEV$M_SPL	Device is spooled.
DEV$M_AVL	Printer is online and available.
Static Bits (Always Set)	
DEV$M_REC	Device is record-oriented.
DEV$M_CCL	Carriage control is enabled.
DEV$M_ODV	Device is capable of output.

[1] Defined by the $DEVDEF macro.

Table 5–2 Device-Dependent Characteristics for Line Printers

Value[1]	Meaning
LP$M_CR	Printer requires carriage return. (See Section 5.2.1).
LP$M_FALLBACK	Printer translates multinational characters to a seven-bit equivalent representation if possible. Otherwise, an underscore character (_) replaces the character. LPM$M_FALLBACK has no effect on physical block operations. See Appendix B for a list of multinational characters.
LP$M_LOWER	Printer can print lowercase characters. If this value is not set, all lowercase characters are converted to uppercase when output. (LP$M_LOWER has no effect on write physical block operations.)
LP$M_MECHFORM	Printer has mechanical form feed. This characteristic is used when variable form length is required, for example, in check printing. Driver sends ASCII form feed (decimal 12). Otherwise, multiple line feeds are generated. The page length determines the number of line feeds.
LP$M_PASSALL	All output data is in binary (no data interpretation occurs). Data termination occurs when the buffer is full (default buffer size is 132 bytes). Character formatting is disabled.
LP$M_PRINTALL	All printing and nonprinting characters are transferred to the printer, while character formatting remains enabled.
LP$M_TAB	Printer enables tab expansion.
LP$M_TRUNCATE	Printer truncates records that are larger than the carriage width.
LP$M_WRAP	Printer wraps records that are larger than the carriage width. If a string of text is longer than the width specified in the second longword, the string is continued on the next line.

[1] Defined by the $LPDEF macro.

5.4 Line Printer Function Codes

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

5.4.1 Write

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

The write functions and their QIO function codes are:

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

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

- P1—the starting virtual address of the buffer that is to be written
- P2—the number of bytes that are to be written
- P3 (ignored)
- P4—carriage control specifier except for write physical block operations (Write function carriage control is described in Section 5.4.1.1.)

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

In write virtual block and write logical block operations, the buffer specified by P1 and P2 is formatted for the selected line printer and includes the carriage control information specified by P4. The default buffer size is 132 bytes.

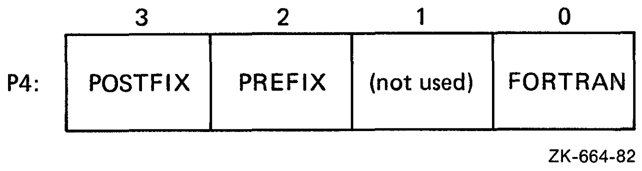
If the printer is not set spooled, write virtual block and write logical block operations perform the same function. If the printer is set spooled, a write logical block function queues the I/O to the printer, and a write virtual block function queues the I/O to the intermediate device, usually a disk.

All lowercase characters are converted to uppercase if the characteristics of the selected printer do not include LP$M_LOWER. (This does not apply to write physical block operations.)

Multiple line feeds are generated for form feeds only if the printer does not have a mechanical form feed (LP$M_MECHFORM) characteristic. The number of line feeds generated depends on the current page position and the page length.

Section 5.2.1 describes character formatting in greater detail.

Figure 5–1 P4 Carriage Control Specifier



5.4.1.1 Write Function Carriage Control

The P4 argument is a longword that specifies carriage control. Carriage control determines the next printing position on the line printer. (P4 is ignored in a write physical block operation.) Figure 5–1 shows the P4 longword format.

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

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

```
Prefix carriage control - Print - Postfix carriage control
```

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

Table 5–3 Write Function Carriage Control (FORTRAN: byte 0 not equal to 0)

Byte 0 Value (hexadecimal)	ASCII Character	Meaning
20	(space)	Single-space carriage control (Sequence: newline[1] , print buffer contents, return)
30	0	Double-space carriage control (Sequence: newline, newline, print buffer contents, return)
31	1	Page eject carriage control (Sequence: form feed, print buffer contents, return)
2B	+	Overprint carriage control; allows double printing for emphasis or for special effects (Sequence: print buffer contents, return)

[1] A newline is a carriage return followed by a line feed.

Table 5–3 (Cont.) Write Function Carriage Control (FORTRAN: byte 0 not equal to 0)

Byte 0 Value (hexadecimal)	ASCII Character	Meaning
24	$	Prompt carriage control (Sequence: newline, print buffer contents)
All other values		Same as ASCII space character: single-space carriage control

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

Prefix/Postfix Bytes (Hexadecimal)

Bit 7	Bits 0–6	Meaning
0	0	No carriage control is specified, that is, NULL.
0	1–7F	Bits 0 through 6 are a count of newlines (carriage return followed by a line feed).

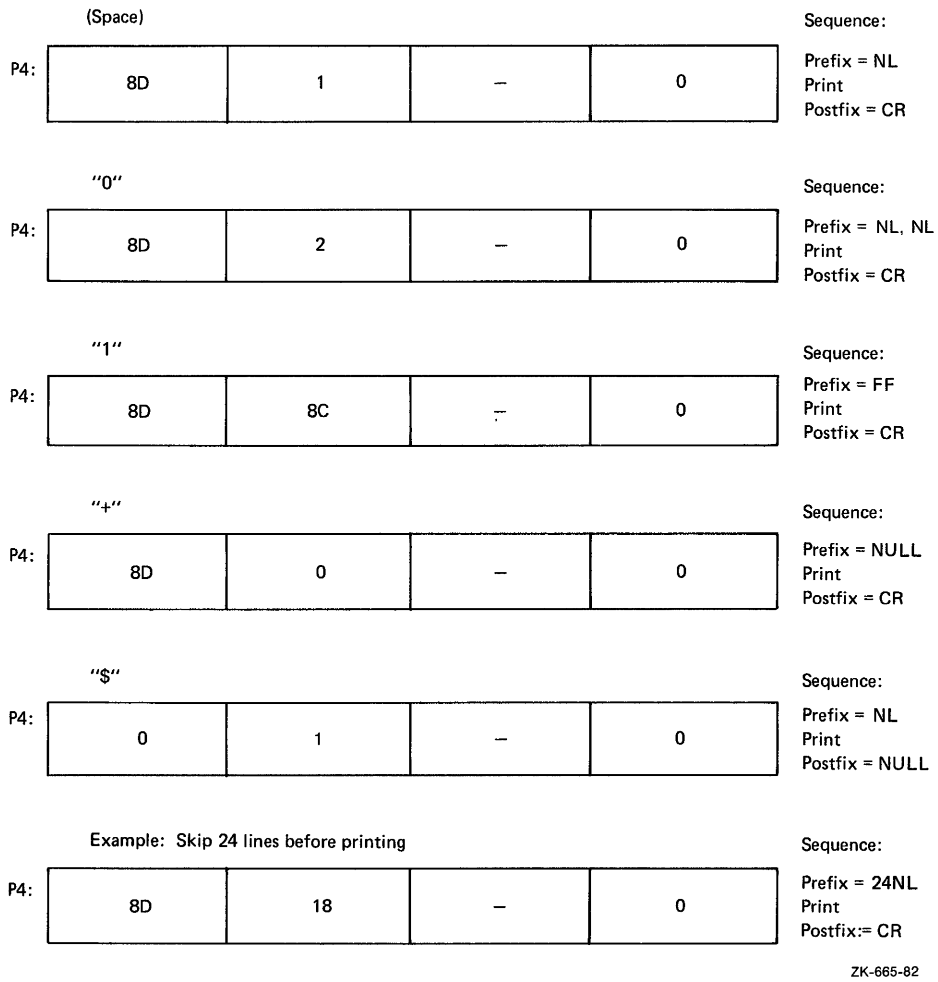
Bit 7	Bit 6	Bit 5	Bits 0–4	Meaning
1	0	0	1–1F	Output the single ASCII control character specified by the configuration of bits 0 through 4 (seven-bit character set).
1	1	0	1–1F	Output the single ASCII control character specified by the configuration of bits 0 through 4, which are translated as ASCII characters 128 through 159 (eight-bit character set; see Appendix B).

Figure 5–2 shows the prefix and postfix hexadecimal coding that produces the carriage control functions listed in Table 5–3. Prefix and postfix coding provides an alternative way to achieve these controls.

In the first example, the prefix/postfix hexadecimal coding for a single-space carriage control (newline, 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)
```

Figure 5–2 Write Function Carriage Control (Prefix and Postfix Coding)


(Space)
Sequence:
P4:
8D
1
–
0
Prefix = NL
Print
Postfix = CR
"0"
Sequence:
P4:
8D
2
–
0
Prefix = NL, NL
Print
Postfix = CR
"1"
Sequence:
P4:
8D
8C
–
0
Prefix = FF
Print
Postfix = CR
"+"
Sequence:
P4:
8D
0
–
0
Prefix = NULL
Print
Postfix = CR
"$"
Sequence:
P4:
0
1
–
0
Prefix = NL
Print
Postfix = NULL
Example: Skip 24 lines before printing
Sequence:
P4:
8D
18
–
0
Prefix = 24NL
Print
Postfix:= CR
ZK-665-82

5.4.2 Sense Printer Mode

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

5.4.3 Set Mode

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

- IO$_SETMODE
- IO$_SETCHAR

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

P1—the address of a characteristics buffer

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

Figure 5–3 Set Mode Buffer

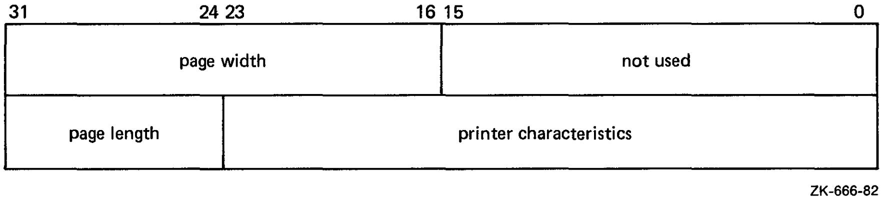
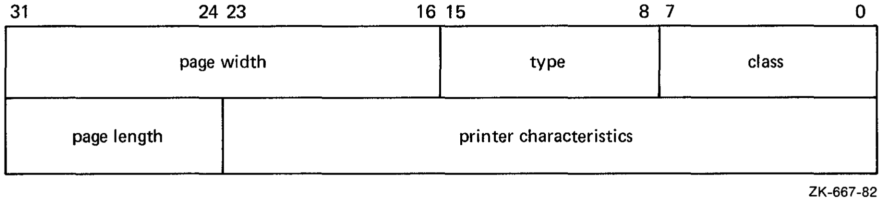

31
24 23
16 15
0
page width
not used
page length
printer characteristics
ZK-666-82

Figure 5–4 Set Characteristics Buffer


31
24 23
16 15
8 7
0
page width
type
class
page length
printer characteristics
ZK-667-82

In the buffer, the device class is DC$_LP. The printer type is a value that corresponds to the printer: DT$_LP27 or DT$_LA11. The type can be changed by the IO$_SETCHAR function. The page width is a value in the range of 0 through 255 on a DMF32 controller and 0 through 65535 on an LP11 or DMB32 controller.

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

Application programs that change specific line printer characteristics should perform the following steps:

1. Use the IO$_SENSEMODE function to read the current characteristics.
2. Modify the characteristics.
3. Use the set mode function to write back the results.

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

5.5 I/O Status Block

The I/O status blocks (IOSB) for the write and set mode I/O functions are shown in Figures 5–5 and 5–6. Appendix A lists the status returns for these functions. (The *VAX/VMS System Messages and Recovery Procedures Reference Manual* provides explanations and suggested user actions for these returns.)

Figure 5–5 IOSB Contents — Write Function

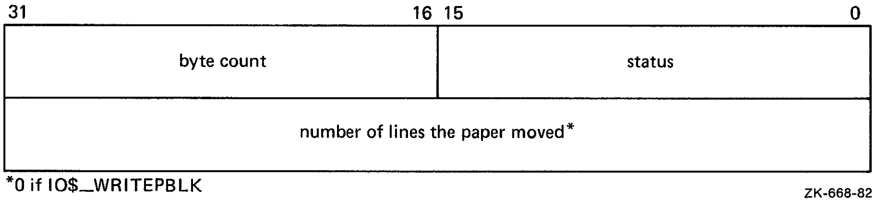
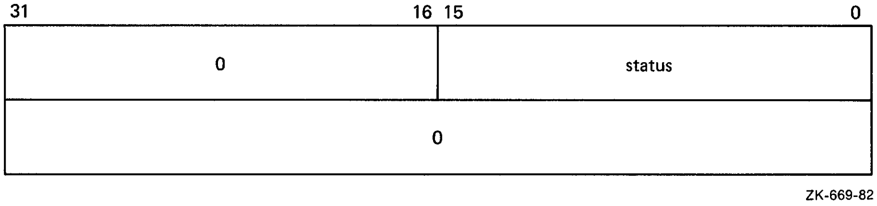

31
16 15
0
byte count
status
number of lines the paper moved*
*0 if IO$_WRITEPBLK
ZK-668-82

Figure 5–6 IOSB Contents — Set Mode Function


31
16 15
0
0
status
0
ZK-669-82

5.6 Programming Example

The following sample program (Example 5–1) is an example of I/O to the line printer that shows how to use the different carriage control formats. This program prints out the contents of the output buffer (OUT_BUFFER) 10 times using 10 different carriage control formats. The formats are held in location OUTPUT_FORMAT.

Example 5–1 Line Printer Program Example

```
; *********************************************************************
;
        .TITLE  LINE PRINTER PROGRAMMING EXAMPLE
        .IDENT  /01/

;
; Define necessary symbols
;
        $IODEF                          ;Define I/O function codes

;
; Allocate storage for the necessary data structures
;
; Allocate output buffer and fill with required output text
;
OUT_BUFFER:
        .ASCII  "VAX_PRINTER_EXAMPLE"
OUT_BUFFER_SIZE=.-OUT_BUFFER            ;Define size of output string

;
; Allocate device name string and descriptor
;
DEVICE_DESCR:                           ;
        .LONG   20$-10$                 ;Length of name string
        .LONG   10$                     ;Address of name string
10$:    .ASCII  /LINE_PRINTER/          ;Name string of output device
20$:                                    ;Reference label to calculate
;                                       ;length

;
; Allocate space to store assigned channel number
;
DEVICE_CHANNEL:                         ;
        .BLKW   1                       ;Channel number

;
; Now set up the carriage control formats
;
OUTPUT_FORMAT:                          ;
        .BYTE   0,0,0,0                 ;No carriage control
        .BYTE   32,0,0,0                ;Blank=LF+...TEXT...+CR
        .BYTE   48,0,0,0                ;Zero=LF+LF+...TEXT...+CR
        .BYTE   49,0,0,0                ;One=FF+...TEXT...+CR
        .BYTE   43,0,0,0                ;Plus=Overprint...+CR
        .BYTE   36,0,0,0                ;Dollar=LF+TEXT(Prompt)

;
; Now the prefix-postfix carriage control formats
;
        .BYTE   0,0,1,141               ;LF+...TEXT...+CR
        .BYTE   0,0,24,141              ;24LF+...TEXT...+CR
        .BYTE   0,0,2,141               ;LF+LF+...TEXT...+CR
        .BYTE   0,0,140,141             ;FF+...TEXT...+CR
```

(Continued on next page)

Example 5–1 (Cont.) Line Printer Program Example

```
;
; *******************************************************************
;
;                       Start Program
;
; *******************************************************************

;
; The program assigns a channel to the output device, sets up a loop
; count for the number of times it wishes to print, and performs ten
; QIO and wait ($QIOW) system service requests.  The channel is then
; deassigned.
;
        .ENTRY  PRINTER_EXAMPLE,^M<R2,R3>  ;Program starting address

;
; First, assign a channel to the output device
;
        $ASSIGN_S DEVNAM=DEVICE_DESCR,- ;Assign a channel to printer
                CHAN=DEVICE_CHANNEL     ;
        BLBC    R0,50$                  ;If low bit = 0, assign failure
        MOVL    #11,R3                  ;Set up loop count
        MOVAL   OUTPUT_FORMAT,R2        ;Set up o/p format address
                                        ;in R2

;
; Start of printing loop
;
30$:    $QIOW_S CHAN=DEVICE_CHANNEL,-   ;Print on device channel
                FUNC=#IO$_WRITEVBLK,-   ;I/O function is write virtual
                P1=OUT_BUFFER,-         ;Address of output buffer
                P2=#OUT_BUFFER_SIZE,-   ;Size of buffer to print
                P4=(R2)+                ;Format control in R2
                                        ;will autoincrement
        BLBC    R0,40$                  ;If low bit = 0, I/O failure
        SOBGTR  R3,30$                  ;Branch if not finished
40$:    $DASSGN_S CHAN=DEVICE_CHANNEL   ;Deassign channel
50$:    RET                             ;Return
        .END    PRINTER_EXAMPLE
```

6 Magnetic Tape Drivers

This section describes the use of the VAX/VMS magnetic tape drivers. These drivers support the devices listed in Table 6–1 and detailed in Section 6.1.

Table 6–1 Supported Magnetic Tape Devices

Controller	Drive[1]	Code	No. of Tracks	Recording Density (bpi)	Tape Speed (ips)	Max. Data Transfer Rate (bytes per second)	Recording Method[2]
TS11	TS04	MS	9	1600	45	72,000	PE
TM03	TE16	MT	9	800 or 1600	45	36,000 (for 800 bpi); 72,000 (for 1600 bpi)	NRZI or PE
	TU45	MT	9	800 or 1600	75	60,000 (for 800 bpi); 120,000 (for 1600 bpi)	NRZI or PE
	TU77	MT	9	800 or 1600	125	100,000 (for 800 bpi); 200,000 (for 1600 bpi)	NRZI or PE
TM78	TU78	MF	9	1600 or 6250	125	200,000 (for 1600 bpi): 781,250 (for 6250 bpi)	PE or GCR
[3]	TU80	MS	9	1600	25 or 100	160,000	PE
[3]	TU81 TU81 -Plus	MU	9	1600 or 6250	25 or 75	120,000 (for 1600 bpi); 468,750 (for 6250 bpi)	PE or GCR
HSC50	TA81	MU	9	1600 or 6250	25 or 75	120,000 (for 1600 bpi); 468,750 (for 6250 bpi)	PE or GCR
	TA78	MF	9	1600 or 6250	125	200,000 (for 1600 bpi); 781,250 (for 6250 bpi)	PE or GCR
TUK50 TQK50	TK50	MU	22[4]	6666	75	45,000	MFM

[1] The TK50, TU81, TU81-Plus, TA78, TU78, and TA81 are tape mass storage control protocol (TMSCP) drives

[2] NRZI = non-return-to-zero-inverted; PE = phase encoded; GCR = group-coded recording; MFM = Modified Frequency Modulation

[3] Has a self-contained controller

[4] Each track written separately—not in parallel

6.1 Supported Magnetic Tape Controllers

The following sections describe the VAX/VMS magnetic tape controllers in greater detail.

6.1.1 TM03 Magnetic Tape Controller

The TM03 magnetic tape controller supports up to a total of eight TE16, TU45, or TU77 tape drives. The difference among these dual-density (800 or 1600 bpi) drives is one of speed: the TE16, TU45, and TU77 read and write data at 45, 75, and 125 inches per second, respectively. Each drive can hold one 2400-foot, 9-track reel with a capacity of approximately 40 million characters. The TM03 controller is connected to the MASSBUS through a MASSBUS adapter.

6.1.2 TS11 Magnetic Tape Controller

The TS11 magnetic tape controller connects to the UNIBUS through a UNIBUS adapter and supports one TS04 tape drive. The TS11/TS04 is a single-density tape system that supports 1600-bpi, phase-encoded recording.

6.1.3 TM78 Magnetic Tape Controller

The TM78 magnetic tape controller supports up to four TU78 tape drives. These high-performance, dual-density drives (1600 or 6250 bpi) operate at 125 inches per second (ips) using a 2400-foot reel of tape with a capacity of approximately 146 million characters when recorded in the GCR (6250 bpi) mode. The TM78 controller is connected to the MASSBUS through a MASSBUS adapter.

6.1.4 TU80 Magnetic Tape Subsystem

The TU80 is a single-density, dual-speed (25 or 100 ips) magnetic tape subsystem that uses streaming tape technology (see Section 6.2.4). It supports one drive per subsystem. The TU80 connects to the UNIBUS through a UNIBUS adapter and completely emulates the TS11 magnetic tape controller.

6.1.5 TU81 and TA81 Magnetic Tape Subsystems

The TU81 and the TA81 are high-performanace, dual-density (1600 or 6250 bpi), dual-speed (25 or 75 ips) magnetic tape subsystems that use streaming tape technology (see Section 6.2.4). The TU81 connects to the UNIBUS through a UNIBUS adapter. The TA81 attaches to an HSC50 controller. Both drives are managed with the tape mass storage control protocol (TMSCP).

6.1.6 TK50 Cartridge Tape System

The TK50 is a 5.24-inch, 95-megabyte cartridge tape that uses streaming tape technology (see Section 6.2.4). The TK50 records data serially on 22 tracks using serpentine recording, rather than on separate (parallel) tracks. The TQK50 is a dual-height Q-bus controller for the TK50 tape drive. The TUK50 is a UNIBUS controller for the same drive. Both the TQK50 and the TUK50 are TMSCP devices. Only one drive is supported per controller.

6.2 Driver Features and Capabilities

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

- Multiple master adapters and slave formatters
- Different types of devices on a single MASSBUS adapter; for example, an RP05 disk and a TM03 tape formatter
- Reverse read function (except for the TK50)
- Reverse data check function (except for TS11 and TK50)
- Data checks on a per-request, per-file, and/or per-volume basis (except for TS11)
- Full recovery from power failure for online drives with volumes mounted, including repositioning by the driver
- Extensive error recovery algorithms; for example, non-return-to-zero-inverted (NRZI) error correction
- Logging of device errors in a file that may be displayed by field service or customer personnel
- 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.

6.2.1 Master Adapters and Slave Formatters

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

VAX/VMS also supports the use of multiple slave formatters per master adapter on a system. For example, more than one TM03 or TM78 magnetic tape formatter per MBA can be used on a system. A slave formatter accepts data and commands from a master adapter and directs the operation of one or more slave drives. The TM03 and the TM78 are slave formatters. The TE16, TU45, TU77, and TU78 magnetic tape drives are slave drives.

6.2.2 Data Check

A data check is made after successful completion of an I/O operation to compare the data in memory with that on the tape. After a write or read (forward) operation, the tape drive 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. Except on TS04 and TU80 drives, magnetic tape drivers support data checks at three levels:

- Per request—Users can specify the data check function modifier (IO$M_DATACHECK) on a read logical block, write logical block, read virtual block, write virtual block, read physical block, or write physical block I/O function.
- Per volume—Users can specify the characteristics "data check all reads" and "data check all writes" when the volume is mounted. The *VAX/VMS DCL Dictionary* describes volume mounting and dismounting. The *VAX/VMS System Services Reference Manual* in the *VAX/VMS System Routines Reference Volume* describes the Mount Volume ($MOUNT) and Dismount Volume ($DISMOU) system services.
- Per file—Users can specify the file attributes "data check on read" or "data check on write." File access attributes are specified when the file is accessed. Section 1 of this manual and the *VAX Record Management Services Reference Manual* in the *VAX/VMS System Routines Reference Volume* both describe file access.

6.2.3 Error Recovery

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

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

The error recovery algorithm uses a combination of these two types of error recovery operations 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 may or may not require operator intervention to reload the drives, depending on, for example, whether the vacuum has been lost on a transport that has vacuum columns. When such operator intervention is required, "device not ready" messages are sent to the operator console to solicit reloading of mounted drives.

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

If a nonfatal controller/drive error occurs, the driver (or the controller, depending on the type of drive) attempts to reexecute the I/O operation up to 16 times before returning a fatal error. The driver repositions the tape before each retry.

The inhibit retry function modifier (IO$M_INHRETRY) inhibits all normal (nonspecial conditions) error recovery. If an error occurs, and the request includes that modifier, the operation is immediately terminated, and the driver returns a failure status. IO$M_INHRETRY has no effect on power failure and timeout recovery.

The driver can write up to 16 extended interrecord gaps during the error recovery for a write operation. For the TE16, TU45, and TU77, writing of these gaps can be suppressed by specifying the inhibit extended interrecord gap function modifier (IO$M_INHEXTGAP). This modifier is ignored for the other magnetic tape drives.

6.2.4 Streaming Tape Systems

Streaming tape systems (TU80, TU81, TU81-Plus, TA81, and TK50) 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.

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, that is, takes a running start, 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. 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, that is, streaming, the drive will reposition 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 the user provides sufficient buffering to ensure continuous tape motion. Some streaming tape drives supported by the VAX/VMS operating system (TU80, TU81, TU81-Plus, and TA81) are dual-speed devices that automatically adjust the tape speed to maximize data throughput and minimize thrashing.

6.3 Device Information

Users can obtain information on all magnetic tape device characteristics by using the Get Device/Volume Information ($GETDVI) system service. (See the *VAX/VMS System Services Reference Manual* in the *VAX/VMS System Routines Reference Volume*.)

$GETDVI returns magnetic tape characteristics when you specify the item codes DVI$_DEVCHAR, DVI$_DEVCHAR2, DVI$_DEVDEPEND, and DVI$_DEVDEPEND2. Tables 6–2, 6–3, and 6–4 list these characteristics. The $DEVDEF macro defines the device-independent characteristics, the $MTDEF macro defines the device-dependent characteristics, and the $MT2DEF macro defines the extended device characteristics. The extended device characteristics apply only to the TU81-Plus.

Table 6–2 Magnetic Tape Device-Independent Characteristics

Characteristic[1]	Meaning
Dynamic Bits (Conditionally Set)	
DEV$M_AVL	Device is online and available.
DEV$M_FOR	Volume is foreign.
DEV$M_MNT	Volume is mounted.
DEV$M_RCK	Perform data check all reads.
DEV$M_WCK	Perform data check all writes.
Static Bits (Always Set)	
DEV$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 6–3 Device-Dependent Information for Tape Devices

Characteristic[1]	Meaning
MT$M_LOST	If set, the current tape position is unknown.
MT$M_HWL	If set, the selected drive is hardware write-locked.
MT$M_EOT	If set, an end-of-tape (EOT) condition was encountered by the last operation to move tape in the forward direction.
MT$M_EOF	If set, a tape mark was encountered by the last operation to move tape.
MT$M_BOT	If set, a beginning-of-tape (BOT) marker was encountered by the last operation to move tape in the reverse direction.

[1] Defined by the $MTDEF macro.

Table 6–3 (Cont.) Device-Dependent Information for Tape Devices

Characteristic[1]	Meaning	
MT$M_PARITY	If set, all data transfers are performed with even parity. If clear (normal case), all data transfers are performed with odd parity. Only non-return-to-zero-inverted recording at 800 bpi can have even parity.	
MT$V_DENSITY MT$S_DENSITY	Specifies the density at which all data transfer operations are performed. Possible density values are:	
	MT$K_GCR_6250	Group-coded recording, 6250 bpi
	MT$K_PE_1600	Phase-encoded recording, 1600 bpi
	MT$K_NRZI_800	Non-return-to-zero-inverted recording, 800 bpi
	MT$K_BLK_833	Cartridge block mode recording[2]
MT$V_FORMAT MT$S_FORMAT	Specifies the format in which all data transfers are performed. A possible format value is:	
	MT$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

Table 6–4 Extended Device Characteristics for Tape Devices

Characteristic[1]	Meaning
MT2$V_WBC_ENABLE	If set, write-back caching is enabled for this unit.
MT2$V_RDC_DISABLE	If set, read caching is disabled for this unit.

[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, and so on.

6.4 Magnetic Tape Function Codes

The VAX/VMS magnetic tape driver can perform logical, virtual, and physical I/O functions. Foreign-mounted devices do not require privilege to perform logical and virtual I/O requests.

Logical and physical I/O functions to magnetic tape devices allow sequential access to volume storage and require only that the requesting process have direct access to the device. The results of logical and physical I/O operations are unpredictable if an ACP is present.

Virtual I/O functions require intervention by an ACP and must be executed in a prescribed order. The normal order is to create and access a file, write information to that file, and deaccess the file. Subsequently, when you access the file, you read the information and then deaccess the file. You can write over the file when the information it contains is no longer useful and the file has expired.

Any number of bytes (from a minimum of 14 to a maximum of 65,535) can be read from or written into a single block by a single request. The number of bytes itself has no effect on the applicable quotas (direct I/O, buffered I/O, and AST). Reading or writing any number of bytes subtracts the same amount from a quota.

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

Table 6–5 lists the logical, virtual, and physical magnetic tape I/O functions and their function codes. These functions are described in more detail in the following paragraphs. Section 1 describes the QIO level interface to the magnetic tape device ACP.

Table 6–5 Magnetic Tape I/O Functions

Function Code and Arguments	Type[1]	Function Modifiers	Function
IO$_CREATE P1,- [P2[,[P3],[P4],- [P5]	V	IO$M_CREATE IO$M_ACCESS	Create a file.
IO$_ACCESS P1,- P2],[P3],[P4],- [P5]	V	IO$M_CREATE IO$M_ACCESS	Search a tape for a specified file and access the file if found and IO$M_ACCESS is set. If the file is not found and IO$M_CREATE is set, create a file at end-of-tape (EOT) marker.

---

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

Table 6–5 (Cont.) Magnetic Tape I/O Functions

Function Code and Arguments	Type[1]	Function Modifiers	Function
IO$_DEACCESS P1,-[P2],[P3],[P4],-[P5]	V		Deaccess a file and, if the file has been written, write out trailer records.
IO$_DSE[2]	P	IO$M_NOWAIT	Erase a prescribed section of the tape.
IO$_MODIFY P1,-[P2],[P3],[P4],-[P5]	V		Write user labels.
IO$_READVBLK P1,P2	V	IO$M_DATACHECK[3] IO$M_INHRETRY IO$M_REVERSE[4]	Read virtual block.
IO$_READLBLK P1,P2	L	IO$M_DATACHECK[3] IO$M_INHRETRY IO$M_REVERSE[4]	Read logical block.
IO$_READPBLK P1,P2	P	IO$M_DATACHECK[3] IO$M_INHRETRY IO$M_REVERSE[4]	Read physical block.
IO$_WRITEVBLK P1,P2	V	IO$M_DATACHECK[3] IO$M_INHRETRY IO$M_INHEXTGAP[7] IO$M_NOWAIT[5]	Write virtual block.
IO$_WRITELBLK P1,P2	L	IO$M_ERASE[6] IO$M_DATACHECK[3] IO$M_INHRETRY IO$M_INHEXTGAP[7] IO$M_NOWAIT[5]	Write logical block.
IO$_WRITEPBLK P1,P2	P	IO$M_ERASE[6] IO$M_DATACHECK[3] IO$M_INHRETRY IO$M_INHEXTGAP[7] IO$M_NOWAIT[5]	Write physical block.
IO$_REWIND	L	IO$M_INHRETRY IO$M_NOWAIT	Reposition tape to the beginning-of-tape (BOT) marker.

---

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

[2] Only for TMSCP drives

[3] Not for TS04 and TU80

[4] Not for TK50

[5] Only for TU81-Plus drives

[6] Takes no arguments; valid only for TMSCP drives

[7] Only for TE16, TU45, and TU77

Table 6–5 (Cont.) Magnetic Tape I/O Functions

Function Code and Arguments	Type[1]	Function Modifiers	Function
IO$_REWINDOFF	L	IO$M_INHRETRY IO$M_NOWAIT	Rewind and unload the tape on the selected drive.
IO$_UNLOAD	L	IO$M_INHRETRY IO$M_NOWAIT	Rewind and unload the tape on the selected drive.
IO$_SKIPFILE P1	L	IO$M_INHRETRY IO$M_NOWAIT[5]	Skip past a specified number of tape marks in either a forward or reverse direction.
IO$_SKIPRECORD P1	L	IO$M_INHRETRY IO$M_NOWAIT[5]	Skip past a specified number of blocks in either a forward or reverse direction.
IO$_WRITEOF	L	IO$M_INHRETRY IO$M_INHEXTGAP[7] IO$M_NOWAIT[5]	Write an extended interrecord gap followed by a tape mark.
IO$_PACKACK	P		Initialize volume valid bit.
IO$_AVAILABLE	P		Clear volume valid bit.
IO$_SENSEMODE [P1]-[P2][8]	L	IO$M_INHRETRY	Sense the tape characteristics and return them in the I/O status block.
IO$_SENSECHAR [P1]-[P2][8]	P	IO$M_INHRETRY	Sense the tape characteristics and return them in the I/O status block.

---

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

[5] Only for TU81-Plus drives

[7] Only for TE16, TU45, and TU77

[8] 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

Table 6–5 (Cont.) Magnetic Tape I/O Functions

Function Code and Arguments	Type[1]	Function Modifiers	Function
IO$_SETMODE P1,-[P2][8]	L		Set tape characteristics for subsequent operations.
IO$_SETCHAR P1,-[P2][8]	P		Set tape characteristics for subsequent operations.
IO$_ACPCONTROL P1,-[P2],[P3],[P4],-[P5]	V	IO$M_DMOUNT	Perform miscellaneous control functions.[9]

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

[8] 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

[9] See Section 1.6.7 for additional information.

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

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

See Section 1 for more information on these functions.

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

- P1—the starting virtual address of the buffer that is to receive the data in the case of a read operation; or, in the case of a write operation, the virtual address of the buffer that is to be written on the tape.
- P2—the length of the buffer specified by P1

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

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

6.4.1 Read

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

The VAX/VMS operating system provides three read function codes:

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

If a read virtual block function is 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 same way as a file-structured device.

Two function-dependent arguments are used with these codes: P1 and P2. These arguments are described in Section 6.4.

If the read function code includes the reverse function modifier (IO$M_REVERSE), the drive reads the tape in the reverse direction instead of the forward direction. IO$M_REVERSE may not be specified for the TK50 drive.

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 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.

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

6.4.2 Write

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

The VAX/VMS operating system provides three write function codes:

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

If a write virtual block function is 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 same way as a file-structured device.

Two function-dependent arguments are used with these codes: P1 and P2. These arguments are described in Section 6.4.

The IO$M_ERASE function modifier can be used with the IO$_WRITELBLK and IO$_WRITEPBLK function codes to erase a user-selected part of a tape. This modifier propagates an erase pattern of all zeros from the current tape position to 10 feet past the EOT position and then rewinds to the BOT marker.

The data check function modifier (IO$M_DATACHECK) can be used with all write functions. If this modifier is specified, a data check operation is performed after the write operation completes. (The drive performs a space reverse between the write and the data check operations.) The driver forces a data check operation when an error occurs during a write operation. This ensures that the data can be reread. A data check operation is also performed if the volume written, or the volume on which the file resides (virtual write), has the characteristic "data check all writes." Furthermore, a data check is performed after a virtual write if the file has the attribute "data check on write." The TS04 and TU80 tape drives do not support the data check function.

If the IO$M_NOWAIT function modifier is specified, write-back caching is enabled on a per command basis. IO$M_NOWAIT is applicable only to TU81-Plus drives.

If the drive performs a write physical block or a write logical block operation, an EOT status is returned if either of the following conditions occurs and no other error condition exists:

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

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

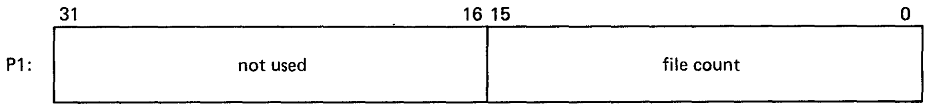
6.4.3 Rewind

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

6.4.4 Skip File

The skip file function skips past a specified number of tape marks in either a forward or reverse direction. A function-dependent argument (P1) is provided to specify the number of tape marks to be skipped, as shown in Figure 6–1. If a positive file count is specified, the tape moves forward; if a negative file count is specified, the tape moves in reverse. (The actual number of files skipped is returned in the I/O status block.)

Figure 6–1 IO$_SKIPFILE Argument



ZK-671-82

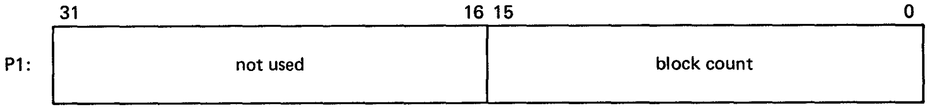
Only tape marks (when the tape moves in either direction) and the BOT marker (when the tape moves in reverse) are counted during a skip file operation. The BOT marker terminates a skip file function in the reverse direction. The end-of-tape (EOT) marker does not terminate a skip file function in either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

A skip file function in the forward direction can also be terminated if two consecutive tape marks are encountered. Section 6.4.5.1 describes this feature.

6.4.5 Skip Record

The skip record function skips past a specified number of physical tape blocks in either a forward or reverse direction. A device/function-dependent argument (P1) specifies the number of blocks to skip, as shown in Figure 6–2. If a positive block count is specified, the tape moves forward; if a negative block count is specified, the tape moves in reverse. (The actual number of blocks skipped is returned in the I/O status block. If a tape mark is detected, the count is the number of blocks skipped, plus 1.)

Figure 6–2 IO$_SKIPRECORD Argument



ZK-672-82

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

A skip record function in the forward direction can also be terminated if the tape was originally positioned between two tape marks. Section 6.4.5.1 describes this feature.

6.4.5.1 Logical End-of-Volume Detection

A skip file or skip record operation is terminated when both of the following conditions exist:

- The tape is mounted foreign.
- Two consecutive tape marks are encountered when the tape moves in the forward direction.

After the operation terminates, the tape remains positioned between the two tape marks that were detected. The I/O status block (IOSB) returns the status SS$_ENDOFVOLUME and the actual number of files (or records) skipped during the operation prior to the detection of the second tape mark. The skip count is returned in the high-order word of the first longword of the IOSB.

Subsequent skip record (or skip file) requests when the tape is positioned between the two tape marks will terminate immediately, 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, the user must employ another I/O function. For example, the IO$_READLBLK function, if issued after receipt of the SS$_ENDOFVOLUME status return, will terminate 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, the user must issue two IO$_READLBLK functions: the first to get the SS$_ENDOFFILE return, the second to position the tape past the third tape mark.

6.4.6 Write End-of-File

The write-end-of-file function writes an extended interrecord gap (of approximately 3 inches for non-return-to-zero-inverted (NRZI) recording and 1.5 inches for phase-encoded (PE) recording) followed by a tape mark. No device/function-dependent arguments are used with IO$_WRITEOF.

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

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

6.4.7 Rewind Offline

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

6.4.8 Unload

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

6.4.9 Sense Tape Mode

The sense tape mode function senses the current device-dependent and extended device characteristics (see Tables 6–3 and 6–4).

The VAX/VMS operating system provides two function codes:

- IO$_SENSEMODE—sense characteristics
- IO$_SENSECHAR—sense characteristics

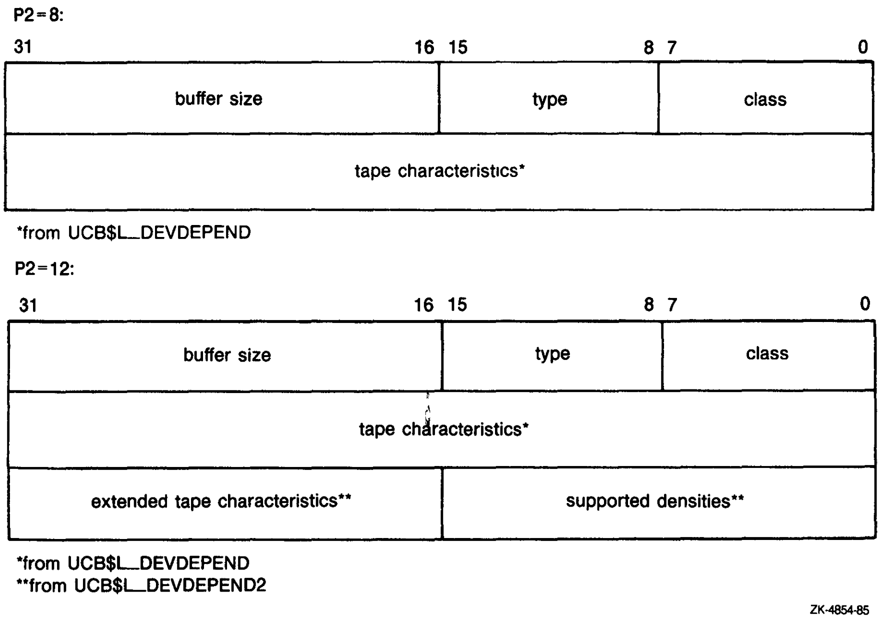
Sense mode requires logical I/O privilege. Sense characteristics requires physical I/O privilege. For TMSCP drives the sense mode function returns magnetic tape information in a user-supplied buffer, which is specified by two function-dependent arguments:

- P1—address of a user-supplied buffer (optional)
- P2—length of user-supplied buffer (optional)

If P1 is not zero, the sense mode buffer returns the tape characteristics. (If P2=8, the second longword of the buffer contains the device-dependent characteristics. If P2=12, the second longword contains the device-dependent characteristics and the third longword contains the tape densities that the drive supports and the extended tape characteristics.) The extended characteristics are identical to the information returned by DVI$_DEVDEPEND2 (see Table 6–4). Figure 6–3 shows the contents of the P1 buffer.

Regardless of whether the P1 buffer is specified, the I/O status block returns the device-dependent characteristics in the second longword (see Figure 6–6). These characteristics are identical to the information returned by DVI$_DEVDEPEND (see Table 6–3 in Section 6.3).

Figure 6–3 Sense Mode P1 Buffer


P2=8:
31
16 15
8 7
0
buffer size
type
class
tape characteristics*
P2=12:
31
16 15
8 7
0
buffer size
type
class
tape characteristics*
extended tape characteristics**
supported densities**
ZK-4854-85

*from UCB$L_DEVDEPEND

*from UCB$L_DEVDEPEND
**from UCB$L_DEVDEPEND2

6.4.10 Set Mode

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

Set mode requires logical I/O privilege. Set characteristics requires physical I/O privilege. Two function codes are provided:

- IO$_SETMODE
- IO$_SETCHAR

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

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

Figure 6–4 shows the P1 characteristics buffer for IO$_SETMODE. Figure 6–5 shows the same buffer for IO$_SETCHAR.

Figure 6–4 Set Mode Characteristics Buffer

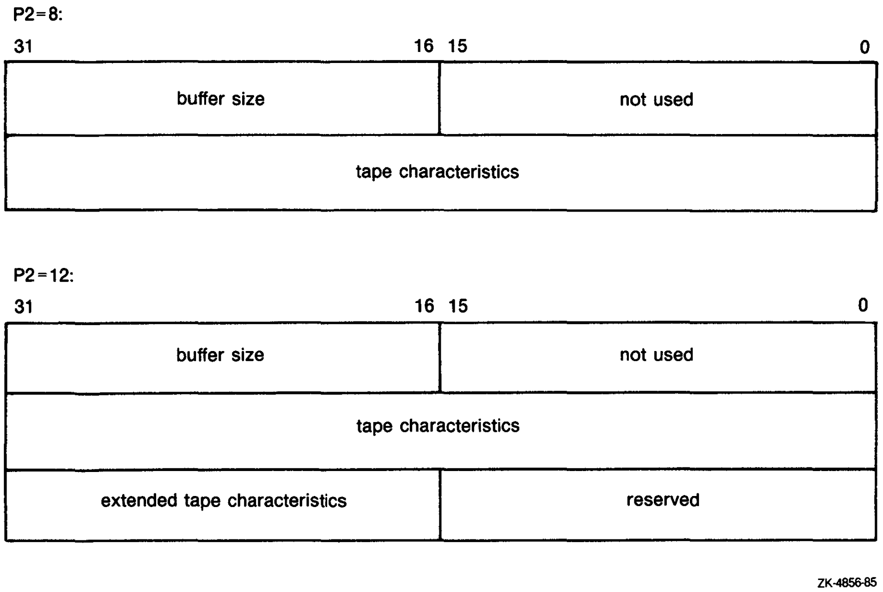
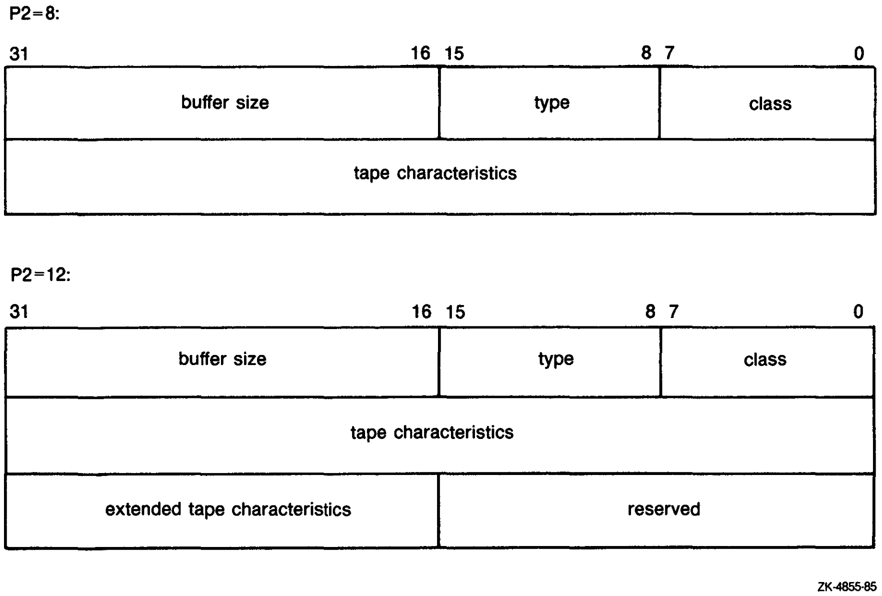

P2=8:
31
16 15
0
buffer size
not used
tape characteristics
P2=12:
31
16 15
0
buffer size
not used
tape characteristics
extended tape characteristics
reserved
ZK-4856-85

Figure 6–5 Set Characteristics Buffer


P2=8:
31
16 15
8 7
0
buffer size
type
class
tape characteristics
P2=12:
31
16 15
8 7
0
buffer size
type
class
tape characteristics
extended tape characteristics
reserved
ZK-4855-85

The first longword of the P1 buffer for the set characteristics function contains information on device class and type, and the buffer size. The device class for tapes is DC$_TAPE.

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

The second longword of the P1 buffer for both the set mode and set characteristics functions contains the tape characteristics. Table 6–6 lists the tape characteristics and their meanings. The $MTDEF macro defines the symbols listed. If P2=12, the third longword contains the extended tape characteristics for TMSCP drives, which are listed in Table 6–7. The extended tape characteristics are defined by the $MT2DEF macro and are identical to the information returned by DVI$_DEVDEPEND2.

Table 6–6 Set Mode and Set Characteristics Magnetic Tape Characteristics

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

Table 6–7 Extended Device Characteristics for Tape Devices

Characteristic[1]	Meaning
MT2$V_WBC_ENABLE	Enable write-back caching on a per unit basis.
MT2$V_RDC_DISABLE	Disable read caching on a per unit basis.

[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 6–2 in Section 6.6:

1. Use the IO$_SENSEMODE function to read the current characteristics.
2. Modify the characteristics.
3. Use the set mode function to write back the results.

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

6.4.11 Data Security Erase

The data security erase function erases all data from the current position of the volume to 10 feet beyond the EOT reflective strip and then rewinds the tape to the BOT marker. It is a physical I/O function and requires the access privilege necessary to perform physical I/O functions. It is applicable only for the TA78, TU78, TA81, TK50, TU81, and TU81-Plus drives. A single 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/function-dependent arguments.

6.4.12 Pack Acknowledge

The pack acknowledge function sets the volume valid bit for all magnetic tape devices. It is a physical I/O function and requires the access privilege to perform physical I/O. A single function code is provided:

- IO$_PACKACK

This function code takes no function-dependent arguments.

IO$_PACKACK must be the first function issued when a volume is placed in a magnetic tape drive. IO$_PACKACK is issued automatically when the DCL commands INITIALIZE or MOUNT are issued.

6.4.13 Available

The available function clears the volume valid bit for all magnetic tape drives, that is, it reverses the function performed by the pack acknowledge function (see Section 6.4.12). No unload function is issued to the drive. A single function code is provided.

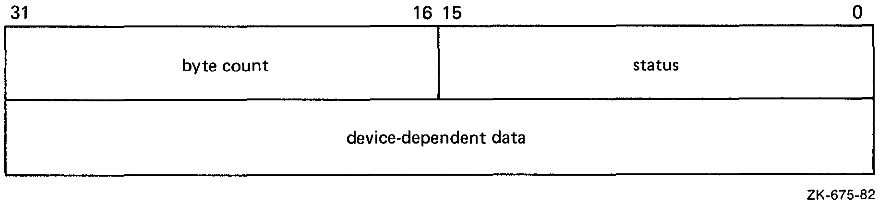
- IO$_AVAILABLE

This function takes no function-dependent arguments.

6.5 I/O Status Block

The I/O status block (IOSB) for QIO functions on magnetic tape devices is shown in Figure 6–6. Appendix A lists the status returns for these functions. (The *VAX/VMS System Messages and Recovery Procedures Reference Manual* provides explanations and suggested user actions for these returns.) Table 6–3 (in Section 6.3) lists the device-dependent data returned in the second longword. The IO$_SENSEMODE function can be used to return that data.

Figure 6–6 IOSB Contents



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 the number of blocks skipped, plus 1.)

6.6 Programming Examples

The following program (Example 6–1) is an example of how data is written to and read from magnetic tape. In the example, QIO operations are performed through the magnetic tape ACP. These operations could have been performed directly on the device using a magnetic tape driver. However, this would have involved additional programming, for example, writing header labels and trailer labels.

Example 6–1 Magnetic Tape Program Example

```
; *******************************************************************
;
        .TITLE  MAGTAPE PROGRAMMING EXAMPLE
        .IDENT  /01/

;
; Define necessary symbols
;
        $FIBDEF                        ;Define file information block
                                       ;symbols
        $IODEF                         ;Define I/O function codes

;
; Allocate storage for the necessary data structures
;

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

;
; Allocate space to store assigned channel number
;
TAPECHAN:                              ;
        .BLKW   1                      ;Tape channel number
;
; Allocate space for the I/O status quadword
;
IOSTATUS:                              ;
        .BLKQ   1                      ;I/O status quadword
;
; Allocate storage for the input/output buffer
;
BUFFER:                                ;
        .REPT   256                    ;Initialize buffer to
        .ASCII  /A/                    ;contain 'A'
        .ENDR                          ;
;
; Now define the file information block (FIB), which the ACP uses
; in accessing and deaccessing the file.  Both the user and the ACP
; supply the information required in the FIB to perform these
; functions.
;
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
```

(Continued on next page)

Example 6–1 (Cont.) Magnetic Tape Program Example

```
;
; **********************************************************************
;
;                          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 - 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 will not allow the reading
; of a file that has been written until one of three events occur:
;       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,^<R3,R4,R5,R6,R7,R8>

;
; First, assign a channel to the tape unit
;
        $ASSIGN_S TAPENAME,TAPECHAN     ;Assign tape unit
        CMPW    #SS$_NORMAL,RO          ;Success?
        BSBW    ERRCHECK                ;Find out

;
; Now create and access the file MYDATA.DAT
;
        $QIOW_S CHAN=TAPECHAN,-         ;Channel is magtape
                FUNC=#IO$_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,RO          ;Success?
        BSBW    ERRCHECK                ;Find out

;
; LOOP1 consists of writing the alphabet to the tape (see previous
; description)
;
        MOVL    #26,R5                  ;Set up loop count
        MOVL    #256,R3                 ;Set up initial byte count
                                        ;in R3
LOOP1:                                  ;Start of loop
        $QIOW_S CHAN=TAPECHAN,-         ;Perform QIOW to tape channel
                FUNC=#IO$_WRITEVBLK,-   ;Function is write virtual
                -                       ;block
                P1=BUFFER,-             ;Buffer address
                P2=R3                   ;Byte count
        CMPW    #SS$_NORMAL,RO          ;Success?
        BSBW    ERRCHECK                ;Find out
```

(Continued on next page)

Example 6–1 (Cont.) Magnetic Tape Program Example

```
;
; Now decrement the byte count in preparation for the next write and
; set up a loop count for updating the character written; LOOP2
; performs the update.
        SUBL2   #2,R3                   ;Decrement byte count for
                                        ;next write
        MOVL    R3,R8                   ;Copy byte count to R8 for
                                        ;LOOP2 count
        MOVAL   BUFFER,R7               ;Get buffer address in R7
LOOP2:  INCB    (R7)+                   ;Increment character
        SOBGTR  R8,LOOP2                ;Until finished
        SOBGTR  R5,LOOP1                ;Repeat LOOP1 until alphabet
                                        ;complete
;
; The alphabet is now complete.  Fall through LOOP1 and update the
; byte count so that it reflects the actual size of the last block
; written to tape.
;
        ADDL2   #2,R3                   ;Update byte count
;
; The tape will now be read but first the program must perform one of
; the three functions described previously before the ACP will allow
; 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
```

(Continued on next page)

Example 6–1 (Cont.) Magnetic Tape Program Example

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

;
; Deassign the channel and exit
;
EXIT:   $DASSGN_S CHAN=TAPECHAN         ;Deassign channel
        RET                             ;Exit

;
; If an error had been detected, a program would normally
; generate some 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
```

The following example (Example 6–2) illustrates the recommended sequence for changing a device characteristic. Simply retrieve the current characteristics using a IO$_SENSEMODE request, set the new characteristics bit(s) and then use IO$_SETMODE to set the new characteristics.

Example 6–2 Device Characteristic Program Example

```
$QIOW_S -                               ; Get current characteristics.
        FUNC    = #IO$_SENSEMODE,-      ; - Sensemode
        CHAN    = CHANNEL,-             ; - Channel
        IOSB    = IO_STATUS,-           ; - IOSB
        P1      = BUFFER,-              ; - User buffer supplied
        P2      = #12                   ; - Buffer length = 12
        .
        .
        .
(Check for errors)
        .
        .
        .
(Set desired characteristics bits)
        .
        .
        .
$QIOW_S -                               ; Set new characteristics.
        FUNC    = #IO$_SETMODE,-        ; - Set Mode
        CHAN    = CHANNEL,-             ; - Channel
        IOSB    = IO_STATUS,-           ; - IOSB
        P1      = BUFFER,-              ; - User buffer address
        P2      = #12                   ; - Buffer length = 12
        .
        .
        .
(Check for errors)
        .
        .
        .
```

The following example (Example 6–3) shows all the various ways of specifying sense mode and set mode, both with and without a user buffer specified, and with user buffers of different lengths.

Example 6–3 Set Mode and Sense Mode Program Example

```
        .PSECT        IMPURE, NOEXE, NOSHR
        $IODEF

DEVICE_NAME:                                  ; Name of device
        .ASCID        /MUAO/                  ;
CHANNEL:                                      ; VMS channel to device
        .WORD        0                        ;
BUFFER: .BLKL        3                        ; Set/Sense characteristics
                                              ;  buffer
IO_STATUS:                                    ; Final I/O status
        .QUAD        0                        ;

        .PSECT        CODE, RD, NOWRT, EXE
        .ENTRY        MAIN,^M<>
        $ASSIGN_S -                           ; Assign a channel to device
                DEVNAM      = DEVICE_NAME,-   ;
                CHAN        = CHANNEL         ;
        BSBW    ERR_CHECK2                    ; Check for errors
        $QIOW_S -                             ; Get current characteristics
                FUNC        = #IO$_SENSEMODE,-; No user buffer supplied
                CHAN        = CHANNEL,-       ;
                IOSB        = IO_STATUS       ;
        BSBW    ERR_CHECK                     ; Check for errors

        $QIOW_S -                             ; Get current characteristics
                FUNC        = #IO$_SENSEMODE,-; User buffer supplied, length
                CHAN        = CHANNEL,-       ;  defaulted
                IOSB        = IO_STATUS,-     ;
                P1          = BUFFER          ;
        BSBW    ERR_CHECK                     ; Check for errors
        $QIOW_S -                             ; Get current characteristics
                FUNC        = #IO$_SENSEMODE,-; User buffer supplied, length
                CHAN        = CHANNEL,-       ;  = 8
                IOSB        = IO_STATUS,-     ;
                P1          = BUFFER,-        ;
                P2          = #8              ;
        BSBW    ERR_CHECK                     ; Check for errors

        $QIOW_S -                             ; Get extended characteristics
                FUNC        = #IO$_SENSEMODE,-; User buffer supplied, length
                CHAN        = CHANNEL,-       ;  = 12
                IOSB        = IO_STATUS,-     ;
                P1        = BUFFER,-          ;
                P2        = #12               ;
        BSBW    ERR_CHECK                     ; Check for errors
        $QIOW_S -                             ; Set new characteristics
                FUNC        = #IO$_SETMODE,-  ; Length defaulted
                CHAN        = CHANNEL,-       ;
                IOSB        = IO_STATUS,-     ;
                P1          = BUFFER          ;
        BSBW    ERR_CHECK                     ; Check for errors
```

(Continued on next page)

Example 6–3 (Cont.) Set Mode and Sense Mode Program Example

```
        $QIOW_S -                               ; Set new characteristics
                FUNC        = #IO$_SETMODE,-    ; Length = 8
                CHAN        = CHANNEL,-         ;
                IOSB        = IO_STATUS,-       ;
                P1          = BUFFER,-          ;
                P2          = #8                ;
        BSBW    ERR_CHECK                       ; Check for errors

        $QIOW_S -                               ; Set extended characteristics
                FUNC        = #IO$_SETMODE,-    ; Length = 12
                CHAN        = CHANNEL,-         ;
                IOSB        = IO_STATUS,-       ;
                P1        = BUFFER,-            ;
                P2        = #12                 ;
        BSBW    ERR_CHECK                       ; Check for errors

        RET

        .ENABLE LSB
ERR_CHECK:
        BLBS    IO_STATUS,ERR_CHECK2            ; Continue if good IOSB
        MOVZWL  IO_STATUS,-(SP)                 ; Otherwise, set up for stop
        BRB     10$                             ; Branch to common code
ERR_CHECK2:
        BLBS    R0,20$                          ; Continue if good status
        PUSHL   R0                              ; Otherwise, set up for stop
10$:    CALLS        #1,G^LIB$STOP              ; Stop execution
20$:    RSB
        .DISABLE LSB

        .END        MAIN
```

7 Mailbox Driver

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

Multiport memory mailboxes function the same as regular mailboxes. However, they can also be used by processes on different processors that are connected to an MA780 multiport memory option.

The *Guide to Programming on VAX/VMS* and the *VAX/VMS System Services Reference Manual* in the *VAX/VMS System Routines Reference Volume* contain additional information on the use of mailboxes.

7.1 Mailbox Operations

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

Table 7–1 Mailbox Read and Write Operations

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

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

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

7.1.1 Creating Mailboxes

To create a mailbox and assign a channel and logical name to it, a process uses the Create Mailbox and Assign Channel ($CREMBX) system service. The system enters the logical name in the job logical name table and gives it an equivalence name of MBAn, where n is a unique unit number.

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

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

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

If sufficient dynamic memory for the mailbox data structure is not available when a mailbox is created, a resource wait will occur if resource wait mode is enabled.

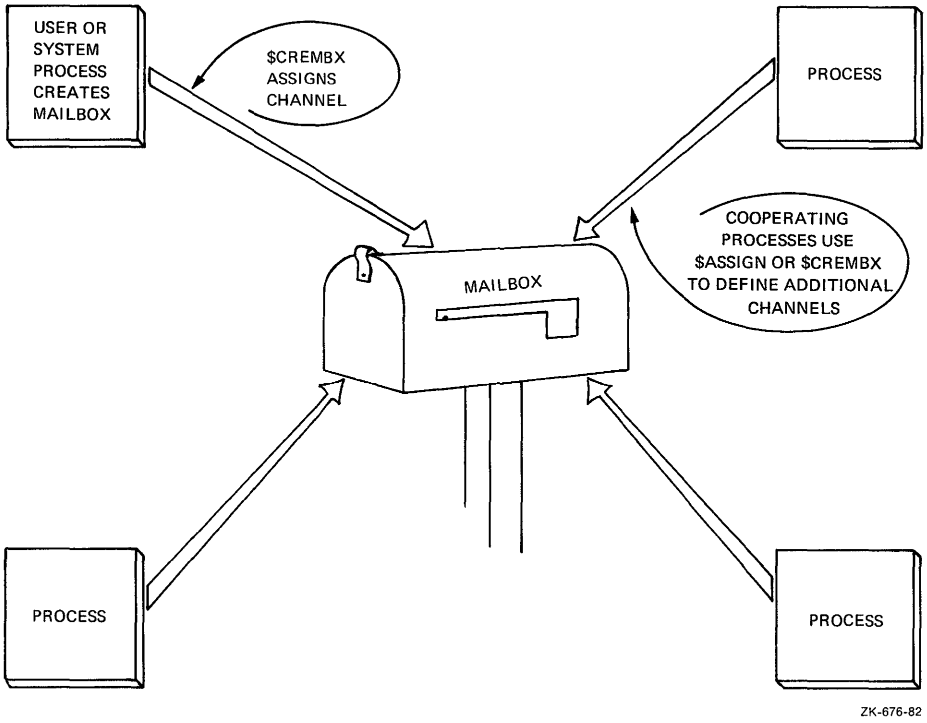
When a mailbox is created, a certain amount of space is specified for buffering messages that have been written to the mailbox but have not yet been read. The **bufquo** argument to the $CREMBX system service specifies this amount or quota. If that argument is omitted, its value defaults to the system generation parameter DEFMBXBUFQUO.

A message written to a mailbox in the absence of an outstanding read request is queued to the mailbox, and the size of the message (the QIO P2 argument) is subtracted from the available buffering space. After the message is read, it is added back to the available buffering space.

If a process attempts to write to a mailbox that is full or has insufficient buffering space, and if the process has resource wait enabled (which is the default case), the process is placed in miscellaneous resource wait mode until sufficient space is available in the mailbox. If resource wait is not enabled, the I/O completes with the status return SS$_MBFULL in the I/O status block (IOSB).

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

Figure 7–1 Multiple Mailbox Channels


USER OR SYSTEM PROCESS CREATES MAILBOX
$CREMBX ASSIGNS CHANNEL
PROCESS
COOPERATING PROCESSES USE $ASSIGN OR $CREMBX TO DEFINE ADDITIONAL CHANNELS
MAILBOX
PROCESS
PROCESS
ZK-676-82

7.1.2 Deleting Mailboxes

As each process finishes using a mailbox, it deassigns the channel using the Deassign I/O Channel ($DASSGN) system service. The channel count is decremented by 1. The system maintains a count of all channels and automatically deletes the mailbox when no more channels are assigned to it (that is, when the channel count reaches 0).

If a mailbox channel is deassigned, all messages sent through that channel are deleted unless the IO$M_NOW function modifier was specified with the write request.

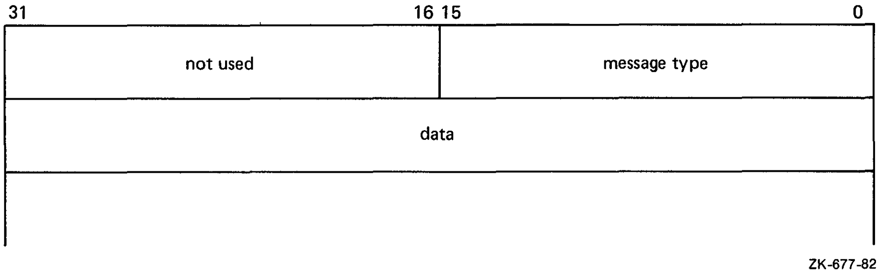
Permanent mailboxes must be explicitly deleted using the Delete Mailbox ($DELMBX) system service. An explicit deletion can occur at any time. However, the mailbox is actually deleted when no processes have channels assigned to it.

When a temporary mailbox is deleted, its message buffer quota is returned to the process that created it. (There is no quota charge made for permanent mailboxes.)

7.1.3 Mailbox Message Format

There is no standardized format for mailbox messages and none is imposed on users. Figure 7–2 shows a typical mailbox message format. Other types of messages can take different formats; for an example, see Figure 8-1 in Section 8.2.4.

Figure 7–2 Typical Mailbox Message Format



7.1.4 Mailbox Protection

Mailboxes (both temporary and permanent) are protected by a code, or mask, that is similar to the code used in protecting volumes. As with volumes, four types of users (defined by UIC) can gain access to a mailbox: SYSTEM, OWNER, GROUP, and WORLD. However, only three types of access—logical I/O, read, and write—are meaningful to users of a mailbox. Thus, when creating a mailbox, the user can specify logical I/O, read, and write access to the mailbox separately for each type of user. Logical I/O access is required for any mailbox operation. The set protection function modifier provides additional control of mailbox access (see Section 7.3.5).

Because temporary mailboxes are customarily used for interprocess communication between cooperating processes with the same group number, they are used more frequently than permanent mailboxes. The logical names of these mailboxes are entered in the group logical name table. Temporary mailboxes thus have two layers of protection. First, easy access to the logical names of temporary mailboxes is granted only to users who have the same group number as the creator of the mailbox; other users have no such easy access. Second, through the protection mask, the creator of the temporary mailbox grants additional security to the mailbox. As a rule, users who are not in the same group as the creator are totally excluded from using the mailbox.

Furthermore, the creator of a temporary mailbox can discriminate between owners and other group members by granting read access and write access to a temporary mailbox—in addition to logical I/O access. For example, owners may be allowed only read access or only write access to the mailbox, but other members of the group may be allowed both read access and write access to the mailbox.

7.2 Device Information

Users can obtain information on mailbox characteristics by using the Get Device/Volume Information ($GETDVI) system service. (See the *VAX/VMS System Services Reference Manual* in the *VAX/VMS System Routines Reference Volume.*)

$GETDVI returns mailbox characteristics when you specify the item code DVI$_ DEVCHAR. Table 7-2 lists these characteristics, which are defined by the $DEVDEF macro.

Table 7–2 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.

7.3 Mailbox Function Codes

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

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

7.3.1 Read

Read mailbox functions are used to obtain messages written by other processes. The VAX/VMS operating system provides three mailbox function codes:

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

Two device/function-dependent arguments are used with these codes:

- P1—the starting virtual address of the buffer that is to receive the message read. If P2 specifies a zero-length buffer, P1 is ignored.
- P2—the size of the buffer in bytes (limited by the maximum message size for the mailbox). A zero-length buffer may be specified. If a message longer than the buffer is read, the alternate success status SS$_BUFFEROVF is returned in the I/O status block. In such cases, the message is truncated to fit the buffer. The driver does not provide a means for recovering the deleted portion of the message.

One function modifier can be specified with a read request:

- IO$M_NOW—complete the I/O operation immediately with no wait for a write request from another process

Figure 7–3 illustrates the read mailbox functions. In this figure process A reads a mailbox message written by process B. As the figure indicates, a mailbox read request requires a corresponding mailbox write request (except in the case of an error). The requests can be made in any sequence; that is, the read request can either precede or follow the write request.

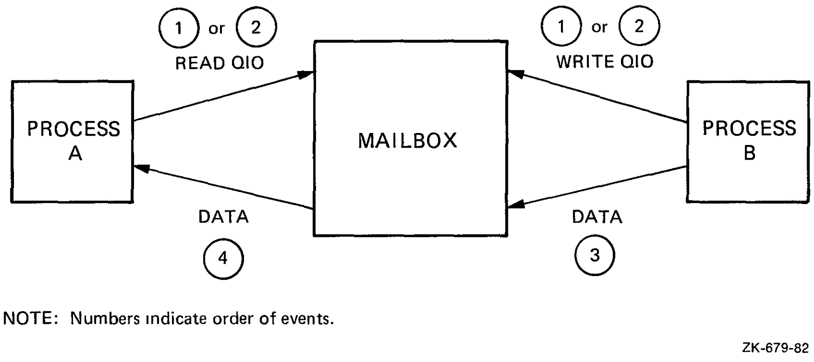
If process A issues a read request before process B issues a write request, either of two things can happen. If process A did not specify the function modifier IO$M_NOW, process A's request is queued before process B issues the write request. When this request occurs, the data is transferred from process B, through the system buffers, to process A to complete the I/O operation.

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

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

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

Figure 7–3 Read Mailbox



7.3.2 Write

Write mailbox functions are used to transfer data from a process to a mailbox. The VAX/VMS operating system provides three mailbox function codes:

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

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

- P1—the starting virtual address of the buffer that contains the message being written. If P2 specifies a zero-length buffer, P1 is ignored.
- P2—the size of the buffer in bytes (limited by the maximum message size for the mailbox). A zero-length buffer produces a zero-length message to be read by the mailbox reader.

One function modifier can be specified with a write request:

- IO$M_NOW—complete the I/O operation immediately with no wait for another process to read the mailbox message

Figure 7–4 illustrates the write mailbox function. In this figure process A writes a message to be read by process B. As in the read request example, a mailbox write request requires a corresponding mailbox read request (unless an error occurs), and the requests can be made in any sequence.

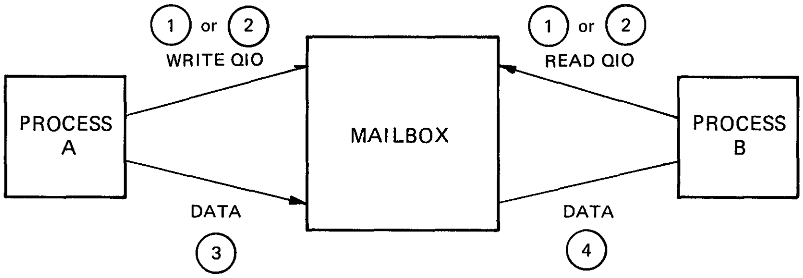
If process A issues a write request before process B issues a read request, either of two things can happen. If process A did not specify the function modifier IO$M_NOW, process A's write request is queued before process B issues a read request. When this request occurs, the data is transferred from 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.

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

Figure 7–4 Write Mailbox



NOTE: Numbers indicate order of events.

ZK-680-82

7.3.3 Write End-of-File Message

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

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

One function modifier can be specified with a write end-of-file request:

- IO$M_NOW—complete the I/O operation immediately with no wait

7.3.4 Set Attention AST

Set attention AST functions are used to specify that an AST be delivered to the requesting process when a cooperating process places an unsolicited read or write request in a designated mailbox. Because the AST only occurs when the read or write request arrives from a cooperating process, the requesting process need not repeatedly check the mailbox status. The user must have both logical I/O and read access to the mailbox prior to performing a set attention AST function.

The VAX/VMS operating system provides two function codes:

- IO$_SETMODE!IO$M_READATTN—read attention AST

- IO$_SETMODE!IO$M_WRTATTN—write attention AST

These function codes take three device/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

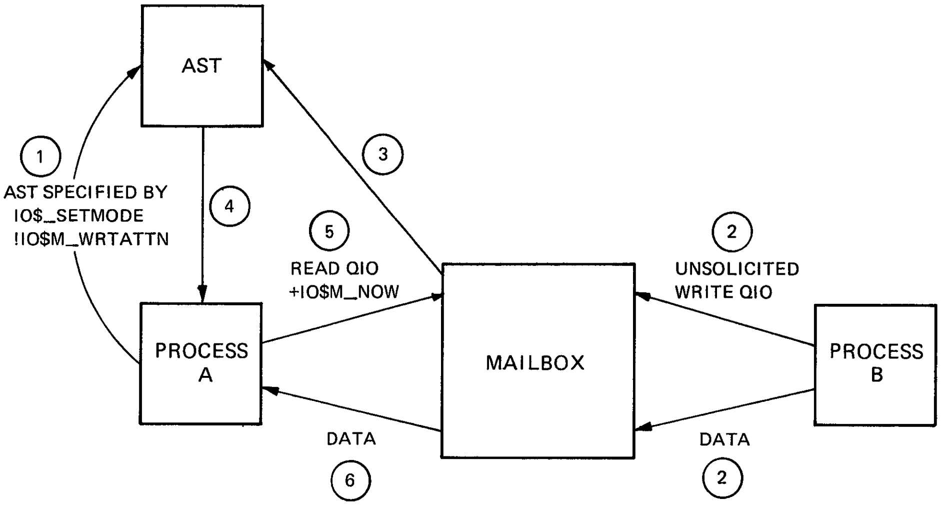
These functions are enabled once only; they must be explicitly reenabled after the AST has been delivered if the user desires notification of the next unsolicited request. Both types of enable functions, and more than one of each type, can be set at the same time. The number of enable functions is limited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

Figure 7–5 illustrates the write attention AST function. In this figure, an AST is set to notify process A when process B sends an unsolicited message.

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

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

Figure 7–5 Write Attention AST (Read Unsolicited Data)



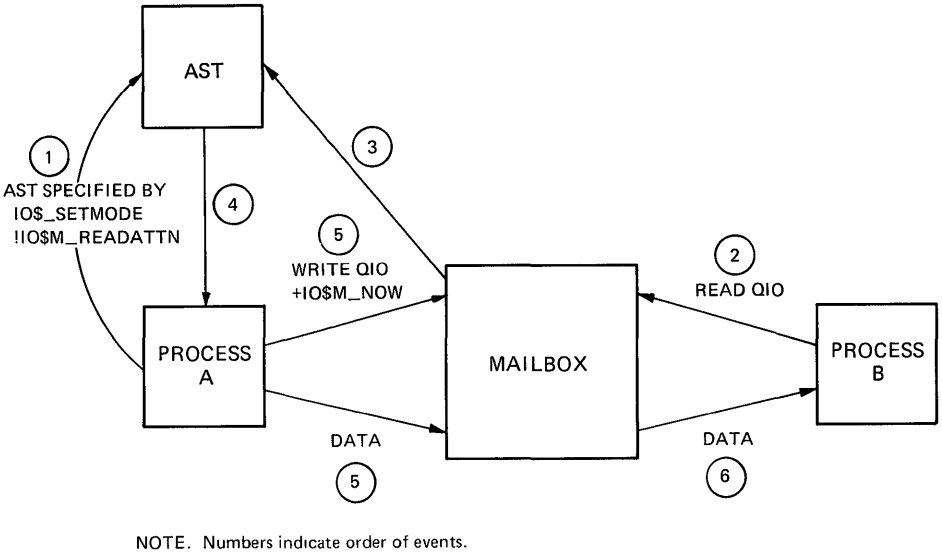
NOTE: Numbers indicate order of events.

Figure 7–6 illustrates the read attention AST function. In this figure, an AST is set to notify process A when process B issues a read request for which no message is available.

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

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

Figure 7–6 Read Attention AST


AST
①
AST SPECIFIED BY
IO$_SETMODE
!IO$M_READATTN
④
③
⑤
WRITE QIO
+IO$M_NOW
②
READ QIO
PROCESS
A
MAILBOX
PROCESS
B
DATA
⑤
DATA
⑥
NOTE. Numbers indicate order of events.
ZK-682-82

7.3.5 Set Protection

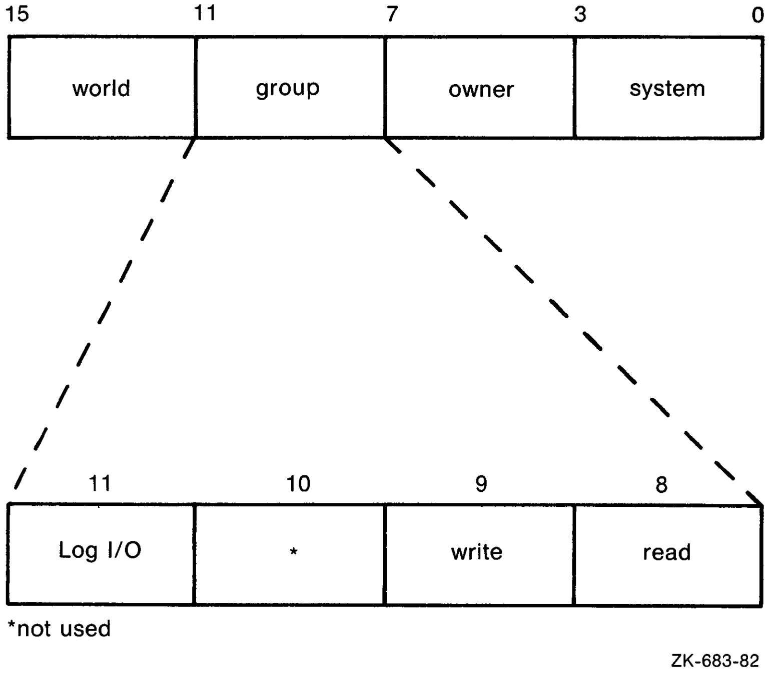
Set protection functions allow the user to set volume protection on a mailbox (see Section 7.1.4). The requester must either be the owner of the mailbox or have BYPASS privilege. The VAX/VMS operating system provides a single function code:

- IO$_SETMODE!IO$M_SETPROT—set protection

This function code takes one device/function-dependent argument:

- P2—a volume protection mask

The protection mask specified by P2 is a 16-bit mask with four bits for each class of owner: SYSTEM, OWNER, GROUP, and WORLD.



Only logical I/O, read, and write functions have meaning for mailboxes. A clear (0) bit implies that access is allowed. If P2 is 0 or unspecified, the mask is set to allow all read, write, and logical operations.

The I/O status block for the set protection function (see Figure 7–9) 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.

The set protection function is typically used when the user wishes to inhibit write access and thus close off input to the mailbox. The mailbox can then be emptied without concern that a write operation might coincide with the read operation.

7.4 I/O Status Block

The I/O status blocks (IOSB) for mailbox read, write, and set protection QIO functions are shown in Figures 7–7, 7–8, and 7–9.

Appendix A lists the I/O status returns for these functions. In addition to these returns, the system services status returns SS$_ACCVIO, SS$_INSFMEM, SS$_MBFULL, SS$_MBTOOSML, and SS$_NOPRIV can be returned in R0. (The *VAX/VMS System Messages and Recovery Procedures Reference Manual* provides explanations and suggested user actions for both types of returns.)

Figure 7–7 IOSB Contents - Read Function

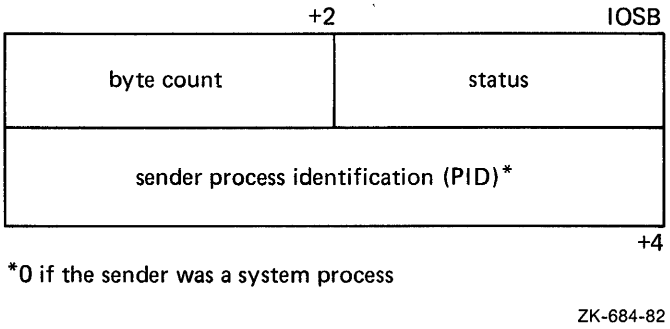

+2
IOSB
byte count
status
sender process identification (PID)*
+4
*0 if the sender was a system process
ZK-684-82

Figure 7–8 IOSB Contents - Write Function

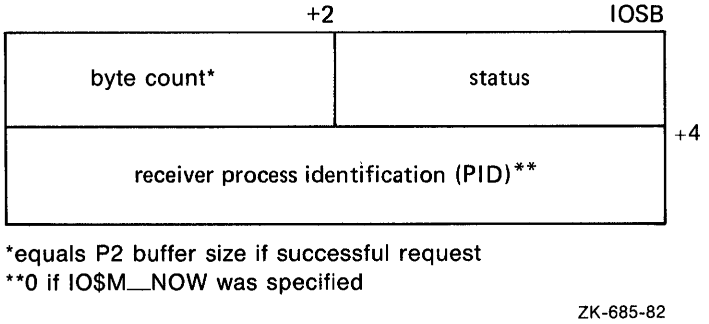
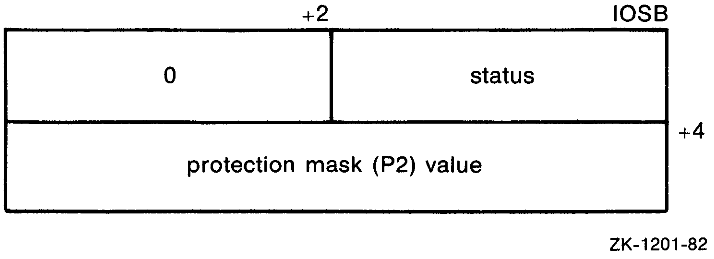

+2
IOSB
byte count*
status
+4
receiver process identification (PID)**
*equals P2 buffer size if successful request
**0 if IO$M_NOW was specified
ZK-685-82

Figure 7–9 IOSB Contents - Set Protection Function


+2
IOSB
0
status
+4
protection mask (P2) value
ZK-1201-82

7.5 Programming Example

The following program (Example 7–1) creates a mailbox and puts mail in it; no matching read is pending on the mailbox. First, the program illustrates that if the function modifier IO$M_NOW is not used when mail is deposited, the write function will wait until a read operation is performed on the mailbox. In this case, IO$M_NOW is specified and the program continues after the mail is left in the mailbox.

Next, the mailbox is read. If there was no mail in the mailbox, the program would wait 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 7–1 Mailbox Driver Program Example

```
; *********************************************************************
;
        .TITLE  MAILBOX DRIVER PROGRAMMING EXAMPLE
        .IDENT  /01/

;
; Define necessary symbols
;
        $IODEF                          ;Define I/O function codes

;
; Allocate storage for necessary data structures
;

;
; Allocate terminal device name string and descriptor
;
DEVICE_DESCR:                           ;
        .LONG   20$-10$                 ;Length of name string
        .LONG   10$                     ;Address of name string
10$:    .ASCII  /TERMINAL/              ;Name string of output device
20$:                                    ;Reference label

;
; Allocate space to store assigned channel number
;
DEVICE_CHANNEL:                         ;
        .BLKW   1                       ;Channel number

;
; Allocate mailbox name string and descriptor
;
MAILBOX_NAME:                           ;
        .LONG   ENDBOX-NAMEBOX          ;Length of name string
        .LONG   NAMEBOX                 ;Address of name string
NAMEBOX: .ASCII /146_MAIN_ST/           ;Name string
ENDBOX:                                 ;Reference label

;
; Allocate space to store assigned channel number
;
MAILBOX_CHANNEL:                        ;
        .BLKW   1                       ;Channel number
```

(Continued on next page)

Example 7–1 (Cont.) Mailbox Driver Program Example

```
;
; 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
;
STATUS:                                 ;
        .QUAD   1                       ;I/O status quadword

;
; *********************************************************************
;
;                      Start Program
;
; *********************************************************************
;

;
; The program first creates a mailbox and assigns a channel to the
; terminal.  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 terminal.
;
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
```

(Continued on next page)

Example 7–1 (Cont.) Mailbox Driver Program Example

```
;
; 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,RO          ;Successful read request?
        BSBW    ERROR_CHECK             ;Find out

;
; The program now determines how much mail was in the mailbox (this
; information is in STATUS+2) and then prints the mailbox message on
; the terminal.
;
        MOVZWL  STATUS+2,R2             ;Byte count into R2
        $QIOW_S FUNC=#IO$_WRITEVBLK,-   ;Write function
                CHAN=DEVICE_CHANNEL,-   ;to the terminal 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 checkng part of the program.  Normally, some kind
; of error recovery would be attempted at this point if an error were
; detected.  However, this example program simply exits.
;
ERROR_CHECK:                            ;
        BNEQ    EXIT                    ;System service failure, exit
        RSB                             ;Otherwise, return
        .END    START
```

8 Terminal Driver

This section describes the use of the VAX/VMS terminal driver. The terminal driver supports the asynchronous, serial line multiplexers listed in Table 8–1 as well as the console terminal.

8.1 Supported Terminal Devices

In addition to the multiplexers listed in Table 8–1, the terminal driver supports serial line interfaces that are included as part of the VAX processor. 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 terminal speed, parity, or any maintenance functions, with the exception of the interface included with the VAX 8200 processor. The VAX/VMS support for this particular interface is included in Table 8–1.

The remote command terminal, used by the DCL command SET HOST, also makes use of the features and capabilities listed in Section 8.2.

Table 8–1 Supported Terminal Devices

Terminal Interface	No. of Lines	Output Silo/DMA	Split Speed	Bus	International Modem Control
DZV11	4	No/No	No	Q-bus	No
DHV11	8	No/Yes	Yes	Q-bus	Full
DZ11	8	No/No	No	UNIBUS	No
DZ32	8	No/No	Limited	UNIBUS	Full
DMF32	8	Yes/Yes (lines 0 and 1)	Yes (lines 0 and 1)	UNIBUS	Yes
DMB32	8	No/Yes	Yes	VAXBI	Full
DHU11	16	Yes/Yes	Yes	UNIBUS	Full
DMZ32	24	Yes/Yes	Yes	UNIBUS	Full
LAT	[1]	No/Yes	[1]	N/A	[1]
VAX 8200 serial lines	4	No/No	No[2]	None	No

[1] Server dependent

[2] The VAX/VMS operating system always supports the first serial line as a console interface. The first serial line, as are the remaining three serial lines, is also supported as a user terminal interface at a maximum speed of 1200 baud in configurations that include a LAT terminal interface and in configurations without other terminal interfaces. However, the VAX/VMS operating system does not support these serial lines as a user terminal interface if a terminal interface other than the LAT is configured.

8.2 Terminal Driver Features and Capabilities

The VAX/VMS terminal driver provides the following features and capabilities:

- Input processing
  - Command line editing and command recall
  - Control characters and special keys
  - Input character validation (read verify)
  - American National Standard (ANSI) escape sequence detection
  - Type-ahead capability
  - Specifiable or default input terminators
  - Special operating modes, such as NOECHO and PASTHRU
- Output processing
  - Efficiency
  - Limited full-duplex operation
  - Formatted or unformatted output
- Dial-up support
  - Modem control
  - Hangup on logging out
  - Preservation of process across hangups
- Miscellaneous
  - Terminal/mailbox interaction
  - Autobaud detection
  - Out-of-band control character handling

8.2.1 Input Processing

The VAX/VMS terminal driver defines a large variety of terminal characteristics and read function modifiers, which provide a wide range of options to an application program. These options allow multiple levels of control over the terminal driver's input process, ranging from the default of command line editing that provides a highly flexible user interface to the PASTHRU mode, which inhibits input process interpretation of data.

8.2.1.1 Command Line Editing and Command Recall

The terminal driver input process defines a bounded set of line editing functions. These functions are available through control keys on all keyboards, and through some special keys on certain keyboards as well. Cursor movement is provided in single-character increments (left arrow or CTRL/D, right arrow or CTRL/F), or in multicharacter increments, to beginning of the line (CTRL/H), or end of the line (CTRL/E). The terminal driver supports both insert character and overstrike character modes. The insert/overstrike mode is the terminal's default characteristic[1] at the beginning of a read operation, but it can be changed dynamically with the toggle insert/overstrike key (CTRL/A). Deletion of characters is supported in both word (CTRL/J or line feed), and to the beginning of the line (CTRL/U) increments.

Use of the terminal driver's editing functions result in several restrictions. These restrictions are:

- The cursor cannot be moved to a previous line after a line wrap.
- A character cannot be inserted if the insertion would force a line wrap or a tab follows the current cursor position.
- A word cannot be deleted at the beginning of a line after a line wrap.
- The line editing function cannot be assigned to other keys.

Command recall, initiated by CTRL/B or the up arrow, returns the last line entered to the command line buffer. At this point, the line can be edited or simply reentered by pressing the RETURN key. DCL extends command recall to the last 20 commands by using the TRM$M_TM_NORECALL modifier to disable the terminal driver's recall mechanism.

Any control key that is not defined by line editing is ignored. For application programs that require control key input but do not perform QIO functions with special read modifiers, the SET TERMINAL/NOLINE_EDIT DCL command restores most of the terminal driver behavior described under VAX/VMS Versions 3.0 through 3.7

8.2.1.2 Control Characters and Special Keys

A control character is a character that controls action at the terminal rather than passing data to a process. An ASCII control character has a code between 0 and 31, plus 125, 126, and 127 (hexadecimal 0 through 1F, plus 7D, 7E, and 7F), that is, all normal control characters plus DELETE and ALTMODE. (Table 1 in Appendix B lists the numeric values for all control characters.) Some control characters are entered at the terminal by simultaneously pressing the CTRL key and a character key, that is, CTRL/x. Other control characters, for example, RETURN, LINE FEED, and ESCAPE, are entered by pressing a single key, that is, RET, LF, and ESC. Table 8–2 lists the VAX/VMS terminal control characters. Control character echo strings (CTRL/C, CTRL/Y, CTRL/O, and CTRL/Z) can be changed on a systemwide basis (see the *VAX/VMS System Manager's Reference Manual*). Several of the control characters do not function as described if the SET TERMINAL/LINE_EDIT DCL command is not specified. See the *VAX/VMS DCL Dictionary* for information on line editing function keys and the SET TERMINAL command.

---

[1] It is suggested that novice users specify overstrike mode.

Table 8–2 Terminal Control Characters

<table>
<tr><th>Control Character</th><th>Meaning</th></tr>
<tr><td>Cancel
(CTRL/C or F6[1])</td><td>Gains the attention of the enabling process if the user program has enabled a CTRL/C AST. If a CTRL/C AST is not enabled, CTRL/C is converted to CTRL/Y (see Section 8.4.3.2).
The terminal performs a newline (carriage return followed by a line feed), echoes CANCEL, and performs another newline. 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:
• 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.</td></tr>
<tr><td>Delete character
(DELETE)</td><td>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.</td></tr>
<tr><td>Delete line
(CTRL/U)</td><td>Purges current input data. When CTRL/U is entered before the end of a read operation, the current input line is deleted. 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.</td></tr>
<tr><td>Delete word
(CTRL/J or F13)
(Line feed)</td><td>Delete word before cursor. Word terminators are all control characters, space, comma, dash, period, and ! " # $ & ' ( ) + @ [\ ] ^ { | ~ / : ; < > = ? (see Appendix B).</td></tr>
</table>

---

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

Table 8–2 (Cont.) Terminal Control Characters

Control Character	Meaning
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.
Interrupt (CTRL/Y)	CTRL/Y is a special interrupt or attention character that is used to gain the attention of the command interpreter for a logged-in process. CTRL/Y can be enabled with an IO$M_CTRLYAST function modifier to a IO$_SETCHAR or IO$_SETMODE function code. The command interpreter's CTRL/Y AST handler always takes precedence over a user program's CTRL/Y AST handler. Entering CTRL/Y results in an AST to an enabled process to signify that the user entered CTRL/Y from the terminal. The terminal performs a newline, echoes INTERRUPT, and performs another newline if the AST and echo are enabled. CTRL/Y is ignored (and not echoed) if the process is not enabled for the AST. Additional consequences of CTRL/Y are: • The type-ahead buffer is flushed. • CTRL/S and CTRL/O are reset. • All queued and in-progress write operations and all in-progress read operations are successfully completed with a 0 transfer count. The status return is SS$_CONTROLY. • The cancel ReGIS escape sequence is sent.
Move cursor left (CTRL/D ←)	Moves the cursor one position to the left.
Move cursor right (CTRL/F →)	Moves the cursor one position to the right.
Move cursor to beginning of line (CTRL/H or F12) (Back space)	Moves the cursor to the beginning of the line.

Table 8–2 (Cont.) Terminal Control Characters

Control Character	Meaning
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 newline is echoed on the terminal, and the current contents of the input buffer are displayed. If the current operation is a read with prompt (IO$_READPROMPT) operation, the current prompt string is also displayed. CTRL/R has no effect if the characteristic TT$M_NOECHO is set.
Restart output (CTRL/Q)	Controls data flow; used by terminals and the driver. Restarts data flow to and from a terminal if previously stopped by CTRL/S. The action occurs immediately with no echo. CTRL/Q is also used to solicit read operations. CTRL/Q is meaningless if the line does not have the characteristic TT$M_TTSYNC, the characteristic TT$M_READSYNC, or is not currently stopped by CTRL/S.
RET (RETURN)	If used during a read (input) operation, RET echoes a newline. 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, that is, multiples of 8, cursor positions. On terminals without this characteristic, enough spaces are output to move the cursor to the next MODULO (8) position.
Status (CTRL/T)	Displays the current time. CTRL/T also displays the current node and user name, the name of the image that is running, and information about system resources that have been used during the current terminal session.
Toggle insert/overstrike (CTRL/A or F14)	Changes current edit mode from insert to overstrike or from overstrike to insert. The default mode (as set with SET TERMINAL/LINE_EDIT) is reset at the beginning of each line.

8.2.1.3 Read Verify

The read verify instructions provided by the terminal driver allow validation of data as each character is entered. Invalid characters are not echoed and terminate the operation. The terminal driver does not support full function field processing. Large data entry applications should use the VAX FMS or VAX TDMS layered products, which support the entire data entry environment. Section 8.4.1.4 describes the supported primitives.

8.2.1.4 Escape and Control Sequences

Escape and control sequences provide additional terminal control not furnished by the control characters and special keys (see Section 8.2.1.2). Escape sequences are strings of two or more characters, beginning with the escape character (decimal 27 or hexadecimal 1B), which indicate that control information follows. Many terminals send and respond to such escape sequences to request special character sets or to indicate the position of a cursor.

The set mode characteristic TT$M_ESCAPE (see Table 8–5) is used to specify that VAX/VMS terminal lines can generate valid escape sequences. Also, the read function modifier IO$M_ESCAPE allows any read operation to terminate on an escape sequence regardless of whether TT$M_ESCAPE is set. If either TT$M_ESCAPE or IO$M_ESCAPE is set, the terminal driver verifies the syntax of the escape sequences. The sequence is always considered a read function terminator and is returned in the read buffer; a read buffer can contain other characters that are not part of an escape sequence, but a complete escape sequence always terminates a read operation. The return information in the read buffer and the I/O status block includes the position and size of the terminating escape sequence in the data record (see Section 8.5).

Any escape sequence received from a terminal is checked for correct syntax. If the syntax is not correct, SS$_BADESCAPE is returned as the status of the I/O. If the escape sequence does not fit in the user buffer, SS$_PARTESCAPE is returned. If SS$_PARTESCAPE is returned, the application program must either use the TRM$_ESCTRMOVR item code, or 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. 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)
```

where

ESC	represents pressing the ESC key, a byte (character) of 1B. This character introduces the escape sequence in a 7-bit environment.
CSI	Control Sequence Introducer, a byte (character) of 9B. This character introduces the escape sequence in a 8-bit environment.
<int>	is an "intermediate character" in the range of 20 to 2F. This range includes the character "space" and 15 punctuation marks. An escape sequence can contain any number of intermediate characters, or none.
<fin>	is a "final character" in the range of 30 to 7E. This range includes uppercase and lowercase letters, numbers, and 13 punctuation marks.

There are three additional escape sequence forms:

```
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)
```

where

ESC	represents pressing the ESC key
[	denotes a control sequence, a byte (character) of 5B
CSI	Control Sequence Introducer
<par>	is a parameter specifier in the range of 30 to 3F
<int>	is an "intermediate character," as defined for escape sequences
<fin>	is a "final character" in the range of 40 to 7E

For example, the position cursor control sequence is ESC [Pl ; Pc H. Pl is the desired line position and Pc is the desired column position.

The user guides for the various terminals list valid escape and control sequences. For example, the *VT100 User Guide* lists the escape and control sequences for VT100 terminals.

Section 8.2.1.2 describes control character functions during escape sequences.

Table 2 in Appendix B lists the valid ANSI and DIGITAL-private escape sequences for terminals that have the TT2$M_ANSICRT, TT2$M_DECCRT, TT2$M_DECCRT2, TT2$M_AVO, TT2$M_EDIT, and TT2$M_BLOCK characteristics (see Table 8–6). Table 2 in Appendix B also lists assumed and selectable ANSI modes and selectable DIGITAL-private modes. Only the names of the escape sequences and modes are listed (for more information see the specific user guide for any of the various terminals). Unless otherwise noted, the operation of escape sequences and modes is identical to the particular terminals that implement these features.

8.2.1.5 Type-Ahead Capability

Input (data received) from a VAX/VMS terminal is always independent of concurrent output (data sent) to a terminal. This capability is called type-ahead. Type-ahead is allowed on all terminals unless explicitly disabled by the set mode characteristic, inhibit type-ahead (TT$M_NOTYPEAHD; see Table 8–5 and Section 8.4.3).

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

Deferring the echo until the read operation is active allows the user process to specify function code modifiers that modify the read operation. These modifiers can include, for example, noecho (IO$M_NOECHO) and convert lowercase characters to uppercase (IO$M_CVTLOW) (see Table 8–7).

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

The action of the driver when the type-ahead buffer fills depends on the set mode characteristic TT$M_HOSTSYNC (see Table 8–5 and Section 8.4.3). If TT$M_HOSTSYNC is not set, CTRL/G (BELL) is returned to inform the user that the type-ahead buffer is full. If characters are entered when the type-ahead buffer is full, the next read operation will complete with a status return of SS$_DATAOVERUN. If TT$M_HOSTSYNC is set, the driver stops input by sending a CTRL/S and the terminal responds by sending no more characters. These warning operations begin eight characters before the type-ahead buffer fills unless the TT2$M_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.

The type-ahead buffer length is variable, with possible values in the range from 0 through 32,767. The length can be set on a systemwide basis through use of the system generation parameter TTY_TYPAHDSZ. Terminal lines that do a large amount of bulk input should use the characteristic TT2$M_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, may take advantage of an optimization in the driver. If a terminal has the characteristic TT2$M_PASTHRU and the read function modifier IO$M_NOECHO is specified, data is placed directly into the read buffer and thereby eliminates the overhead for moving the data from the type-ahead buffer.

8.2.1.6 Line Terminators

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

Terminators are specified by an argument to the QIO request for a read operation. By default, they can be any ASCII control character except FF, VT, LF, TAB, or BS (see Appendix B). If line editing is enabled, the only terminators are CR, CTRL/Z, or an escape sequence. Control keys that do not have an editing function are nonfunctioning keys. If included in the request, the argument is a user-selected group of characters (see Section 8.4.1.2).

All characters are 7-bit ASCII characters unless data is input on an 8-bit terminal (see Section 8.4.1). (The characteristic TT$M_EIGHTBIT determines whether a terminal uses the 7-bit or 8-bit character set; see Table 8–5.) All input characters (except some special keys; see Section 8.2.1.2) are tested against the selected terminator(s). The input is terminated when a match occurs or the user's 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 capability has not been disabled). A bell character is sent when the notification occurs. A notification character other than the bell character may be specified by setting the system generation parameter TTY_AUTOCHAR.

8.2.1.7 Special Operating Modes

The VAX/VMS terminal driver supports many special operating modes for terminal lines. (Tables 8–5 and 8–6 in Section 8.3 list these modes.) All special modes are enabled or disabled by the set mode and set characteristics functions (see Section 8.4.3).

8.2.2 Output Processing

Output handling is designed to be very efficient in the terminal driver. For example, on multiplexers that support both silo and direct memory access (DMA) 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.

8.2.2.1 Duplex Modes

The VAX/VMS terminal driver can execute in either half- or full-duplex mode. These modes describe the terminal driver software, specifically the ordering algorithms used to service read and write requests, not the terminal communication lines.

In half-duplex mode, all read and write requests are inserted onto one queue. The terminal driver removes requests from the head of this queue and executes them one at a time; all requests are executed sequentially in the order in which they were issued.

In full-duplex mode, read requests (and all other requests except write requests) are inserted onto one queue and write requests onto another. The existence of two queues allows the driver to recognize the presence of two requests, for example, 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 or the write—to process next.

In the VAX/VMS terminal driver, write requests normally have priority. A write request can interrupt a current, but inactive, read request. A read request is current when the terminal driver removes that request from the head of the read queue. In a read operation, the read request becomes active when the first input character is received and echoed. Once a read request becomes active, all write requests will be 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, will not inhibit write operations in full-duplex mode.

If a write function that specifies the IO$M_BREAKTHRU modifier is performed, the write will not be blocked—even by an active read operation. IO$M_BREAKTHRU does not change the order in which write operations are queued.

When all I/O requests are issued using the Queue I/O Request and Wait ($QIOW) system service, there can be only one current I/O request at any 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.

8.2.2.2 Formatting of Output

By default, output data is subject to formatting by the terminal driver. This formatting includes actions such as wrapping, tab expansion, uppercase, and fallback conversions. Applications that do not require formatting of data can write with the IO$M_NOFORMAT modifier and thereby reduce overhead. IO$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.

8.2.3 Dial-Up Support

The VAX/VMS operating system supports modem control (for example, Bell 103A, Bell 113, or equivalent) for all supported multiplexers in autoanswer, full-duplex mode. The terminal driver does not support half-duplex operations on modems such as the Bell 202. Also not supported are modems that use circuit 108/1 (connect data set to line signal) in place of the data terminal ready (DTR) signal. Most U.S. and European modems use the data terminal ready signal, which is the signal supported by the VAX/VMS operating system.

8.2.3.1 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 dial-up monitoring period for multiplexers that do not support modem signal transition interrupts (see Table 8–1).

If a line's carrier signal is lost, the driver waits two seconds for the carrier signal to return. (If bit 0 of the system generation parameter TTY_DIALTYPE is set (=1), the driver does not wait. Bit 0 is 0 by default for countries with Bell System standards, but that bit should be set to 1 for countries with Comite Consultatif Internationale (CCITT) standards.) If the carrier signal is not detected during this time, the line is "hung up." The hangup action can signal the owner of the line, through a mailbox message, that the line is no longer in use. (No dial-in message is sent; the unsolicited character message is sufficient when the first available data is received.) The line is not available for a minimum of two seconds after the hangup sequence begins. The hangup sequence is not reversible. If the line hangs up, all enabled CTRL/Y and out-of-band ASTs are delivered; the CTRL/Y AST P2 argument is overwritten with SS$_HANGUP. The I/O operation in progress is canceled, and the status value SS$_HANGUP is returned in the I/O status block. DCL is responsible for process deletion after CTRL/Y is delivered. If the process is suspended, DCL cannot run, and therefore deletion cannot occur, until the process is resumed.

For terminals with the TT$M_MODEM characteristic, TT$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 8.4.3.3).

The VAX/VMS terminal driver default modem protocol meets the requirements of the United States and of European countries. This protocol is capable of working in automatic answer mode and can also perform manually dialed outgoing calls. The protocol supports the requirements of most known international telephone networks. Use of the enhanced modem features is made on multiplexers that support them; processor polling is not necessary. The protocol also functions in a subset mode for the multiplexers that do not support full modem signals (see Table 8–1).

Table 8–3 lists the control and data signals used in a full modem control mode configuration, that is, in a two-way simultaneous, symmetrical transmit mode. Figure 8–1 is a flowchart that shows a typical signal sequence for a terminal operation in this mode. The flowchart shows the states that the modem transition code goes through to detect different types of transitions in modem state. These transitions allow the driver to detect loss of lines that have been idle for several minutes. Modem states do not affect the ability of the system to transmit characters.

Set mode function modifiers are provided to allow a process to activate or deactivate modem control signals (see Section 8.4.3.3).

Bit 1 of the system generation parameter TTY_DIALTYPE enables alternate modem protocol on a systemwide basis. If bit 1 is 0 (the default), the RING signal is not used. If bit 1 is 1, the modem protocol delays setting the DTR signal until the RING signal is detected.

Remote terminal connections have a timeout feature for the security of dial-up lines. If no channel is assigned to the port within 30 seconds, the DTR signal is dropped. Such action prevents an unused terminal from tying up a line. However, there are configurations (such as a printer connected to a remote line) in which the line should not be dropped even though it is not being used interactively. To bypass the 30-second timeout, set the system generation parameter TTY_DIALTYPE to 4. (Note that if TTY_DIALTYPE is equal to 4, all dial-up lines will skip the timeout.)

Table 8–3 Control and Data Signals (Full Modem Mode Configuration)

Signal	Source	MUX[1]	Meaning
Transmitted data (TxD)	Computer	All	The data originated by the computer and transmitted through the modem to one or more remote terminals.
Received data (RxD)	Modem	All	The data generated by the modem in response to telephone line signals received from a remote terminal and transferred to the computer.

---

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

Table 8–3 (Cont.) Control and Data Signals (Full Modem Mode Configuration)

Signal	Source	MUX[1]	Meaning
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 the modem 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 VAX/VMS operating system initiates a 30-second timer. This ensures that the phone line will be disconnected if CARRIER is not detected.
Data channel received line signal detector (CARRIER)	Modem	All	If present (ON condition), CARRIER indicates that the received data channel line signal is within appropriate limits, as specified by the modem. If not present (OFF condition), the received signal is not within appropriate limits.

---

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

Table 8–3 (Cont.) Control and Data Signals (Full Modem Mode Configuration)

Signal	Source	MUX[1]	Meaning
Data terminal ready (DTR)	Computer	All	If present (ON condition), DTR indicates that the computer is ready to operate, prepares the modem to connect to the telephone line, and maintains the connection after it has been made by other means. DTR can be present whenever the computer is ready to transmit or receive data. If DTR is not present (OFF condition), the modem disconnects the modem from the line.
Calling indicator (RING)	Modem	All	Indicates whether a calling signal is being received by the modem. Bit 1 of the system generation parameter TTY_DIALTYPE must be set (=1). If RING is detected, the VAX/VMS 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, and DHV11)

8.2.3.2 Hangup on Logging Out

By default, logging out on a line with modem signals will not break the connection. If TT2$M_HANGUP is set, modem signals are dropped when the process logs out. If TT2$M_MODHANGUP is set, no privilege is required to change the state of TT2$M_HANGUP. By setting TT2M_HANGUP, system managers can prevent nonprivileged users who are not logged in from tying up a dial-in line.

8.2.3.3 Preservation of a Process Across Hangups

Disconnectable terminals allow a connection to a physical terminal line to be broken without losing the job. The following SYSGEN command allows terminals to be disconnectable terminals:

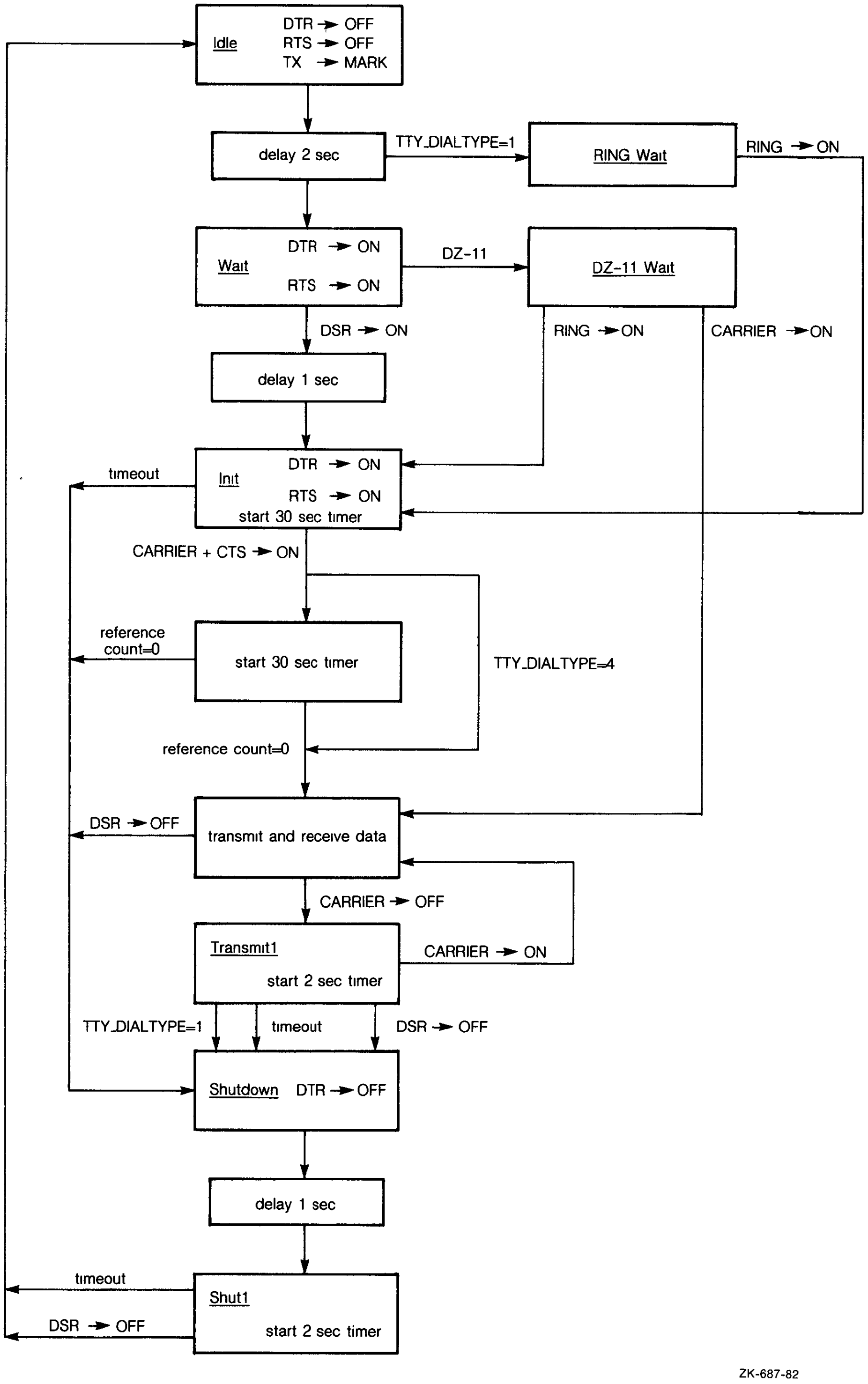
```
SYSGEN> CONNECT VTA0/NOADAPTER/DRIVER=TTDRIVER
```

After this command is entered, a terminal with the TT2$M_DISCONNECT characteristic will log in as VTAn:, rather than 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.

There are four ways in which a terminal can become disconnected:

- Modem signals between the host and the terminal are lost.

Figure 8–1 Modem Control - Two-Way Simultaneous Operation


Idle
DTR → OFF
RTS → OFF
TX → MARK
delay 2 sec
TTY_DIALTYPE=1
RING Wait
RING → ON
Wait
DTR → ON
RTS → ON
DZ-11
DZ-11 Wait
DSR → ON
RING → ON
CARRIER → ON
delay 1 sec
timeout
Init
DTR → ON
RTS → ON
start 30 sec timer
CARRIER + CTS → ON
reference count=0
start 30 sec timer
TTY_DIALTYPE=4
reference count=0
DSR → OFF
transmit and receive data
CARRIER → OFF
Transmit1
start 2 sec timer
CARRIER → ON
TTY_DIALTYPE=1
timeout
DSR → OFF
Shutdown DTR → OFF
delay 1 sec
timeout
Shut1
DSR → OFF
start 2 sec timer
ZK-687-82

- 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.

Once validated as a user, you can connect to a disconnected process in either of the following ways:

- Allow the login process to make the connection.
- Issue the DCL command CONNECT.

8.2.4 Terminal/Mailbox Interaction

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

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

Mailboxes used in this way carry status messages, not terminal data, from the driver to the user program. For example, when data is received from a terminal for which no read request is outstanding (unsolicited data), a message is sent to the associated mailbox to indicate data availability. On receiving this message, the user program reads the channel assigned to the terminal to obtain the data. Messages are sent to mailboxes under the following conditions:

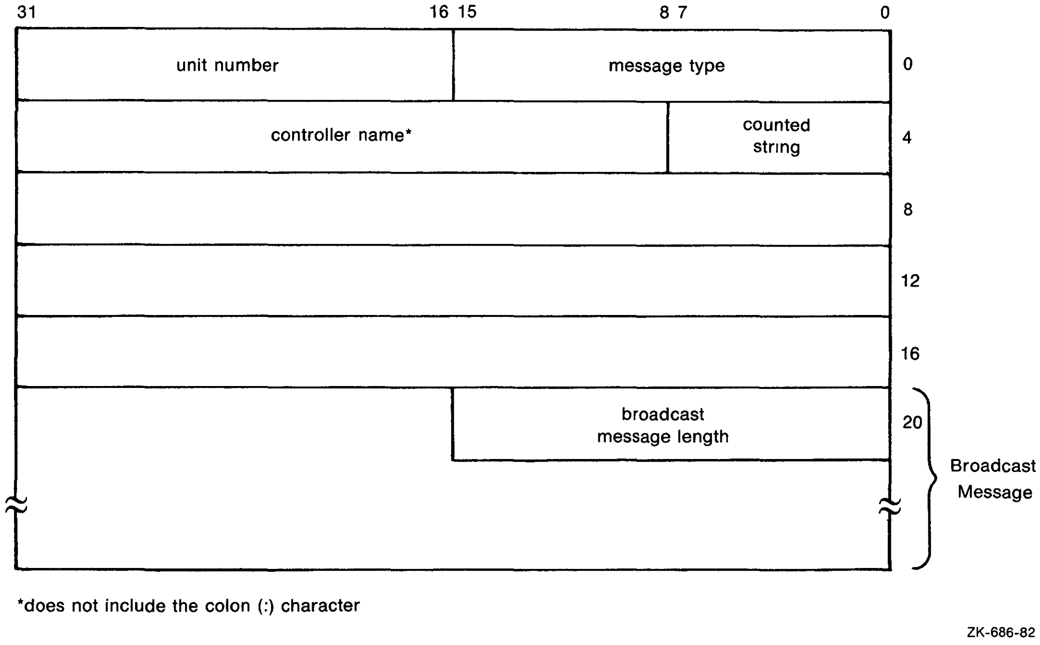
- Unsolicited data in the type-ahead buffer. The use of the associated mailbox can be enabled and disabled as a subfunction of the read and write requests (see Sections 8.4.1 and 8.4.2). (Initially, mailbox messages are enabled on all terminals. This is the default state.) Thus, the user process can enter into a dialogue with the terminal after an unsolicited data message arrives. Then, after the dialogue is over, the user process can reenable the unsolicited data message function on the last I/O exchange. Only one message is sent between read operations.
- Terminal hangup. When a remote line loses the carrier signal, it hangs up; a message is sent to the mailbox. When hangup occurs on lines that have the characteristic TT$M_REMOTE set, the line returns to local mode.
- Broadcast messages. If the characteristic TT2$M_BRDCSTMBX is set, broadcasts sent to a terminal are placed in the mailbox (this is independent of the state of TT$M_NOBRDCST).

Messages placed in the mailbox have the following content and format (see Figure 8–2):

- Message type. The codes MSG$_TRMUNSOLIC (unsolicited data), MSG$_TRMHANGUP (hangup), and MSG$_TRMBRDCST (terminal broadcast) identify the type of message. Message types are identified by the $MSGDEF macro.
- Device unit number to identify the terminal that sent the message.
- Counted string to specify the device name.

- Controller name.
- Message (for broadcasts).

Figure 8–2 Terminal Mailbox Message Format


31
16 15
8 7
0
unit number
message type
0
controller name*
counted string
4
8
12
16
broadcast message length
20
Broadcast Message
*does not include the colon (:) character
ZK-686-82

Interaction with a mailbox associated with a terminal occurs through standard QIO functions and ASTs. Therefore, the process need not have outstanding read requests to an interactive terminal to respond to the arrival of unsolicited data. The process need only respond when the mailbox signals the availability of unsolicited data. Section 8.6 contains an example of mailbox programming.

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

8.2.5 Autobaud Detection

If the user specifies the /AUTOBAUD qualifier with the SET TERMINAL command, automatic baud rate detection is enabled, allowing the terminal baud rate to be set when the user logs 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 may 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 seven 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 eight-bit ASCII code. /NOEIGHT_BIT, which is the default, specifies seven-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 will also work with other combinations, for example, /NOEIGHT_BIT/ NOPARITY.

Table 8–6 describes the terminal characteristic TT2$M_AUTOBAUD, which allows the baud rate to be set automatically at login.

Specifying the /FRAME qualifier with the SET TERMINAL command is not usually recommended. 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 may be necessary to change these values if the terminal is a non-DIGITAL device.

8.2.6 Out-of-Band Control Character Handling

All seven-bit 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 typed. 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.

8.3 Device Information

Users can obtain information on terminal characteristics by using the Get Device/Volume Information ($GETDVI) system service. (See the *VAX/VMS System Services Reference Manual* in the *VAX/VMS System Routines Reference Volume*). The sense mode function provides an alternative means to obtain terminal characteristics; see Section 8.4.4.

$GETDVI returns terminal characteristics when you specify the item codes DVI$_DEVCHAR, DVI$_DEVDEPEND, and DVI$_DEVDEPEND2. Tables 8–4, 8–5 and 8–6 list these characteristics. Terminal characteristics are normally set during system generation to any one of, or a combination of, the values listed in Table 8–5. 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 $DCDEF macro. The device class for terminals is DC$_TERM. The terminal model determines the device type. For example, the device type for the VT240 is DT$_VT240. DVI$_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 8–4 Terminal Device-Independent Characteristics

Characteristic Name[1]	Meaning
DEV$M_AVL	Terminal is online and available.
DEV$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 VAX-DECnet use

[1] Defined by the $DEVDEF macro

Table 8–5 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 eight-bit ASCII character set (see Appendix B). Terminals without this characteristic use the seven-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 8.2.1.4). Escape sequences are validated for syntax.
TT$M_HALFDUP	Terminal is in half-duplex mode (see Section 8.2.2.1). All read and write requests are executed sequentially.
TT$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_LFFILL	Terminal requires fill after the LINEFEED key is pressed. (The fill can be specified by the set mode P4 argument.)

[1] 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 8–5 (Cont.) Terminal Characteristics

Value[1]	Meaning
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.)
TT$M_MBXDSABL	Mailboxes associated with the terminal will not receive notification of unsolicited input or hangup (see Section 8.2.3). This bit can be set by the IO$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$_MODEM is set, the terminal driver automatically handles modem control. If TT$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).
TT$M_NOBRDCST	Terminal will not receive any broadcast messages.
TT$M_NOECHO	Input characters are not echoed on this terminal line (see Section 8.2.1.5).
TT$M_NOTYPEAHD	Data must be solicited by a read operation. Data is lost if received in the absence of an outstanding read request, that is, if it is unsolicited data. Disables type-ahead capability (see Section 8.2.1.5). If this characteristic is set, login attempts on this line are disabled.
TT$M_READSYNC	Read synchronization is enabled. The host explicitly solicits all read operations by issuing a CTRL/Q and terminates the operation by issuing a CTRL/S.
TT$M_REMOTE	Dial-up characteristic is enabled. The terminal returns to local mode when a hangup occurs on the terminal line (see Section 8.2.3). This characteristic cannot be changed; it is only informational.
TT$M_SCOPE	Terminal is a video screen display (CRT terminal), for example, the VT100 or VT240.
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.

[1] 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 8–5 (Cont.) Terminal Characteristics

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

---

[1] 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 8–6 Extended Terminal Characteristics

Value[1]	Meaning
TT2$M_ALTYPEAHD	Alternate type-ahead buffer size is enabled. Use the alternate type-ahead buffer size specified during system generation (see Section 8.2.1.5). If a type-ahead buffer already exists for a terminal line, there is no effect when this characteristic is set for that line. TT2$M_ALTYPEAHD should be set prior to using the terminal, such as in the startup command procedure. Note that the user 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$M_AUTOBAUD	Automatic baud rate detection is enabled. This characteristic allows the baud rate to be set automatically when the user logs in. (The baud rate is set when one or more carriage returns are typed during the login procedure.) Terminals are set to a permanent speed of 9600 baud. If TT2$M_AUTOBAUD is specified, the permanent speed must not be changed while this characteristic is in use on a given terminal line. See Section 8.2.5 for additional information on automatic baud rate detection.

---

[1] 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.

Table 8–6 (Cont.) Extended Terminal Characteristics

Value[1]	Meaning
TT2$M_AVO	Advanced video is enabled. This characteristic provides the terminal with the capability of blink, bold, and flashing fields as well as a full screen of 132 character lines. TT2$M_AVO is set by the SET TERMINAL command. Appendix B lists the valid escape sequences for terminals with the TT2$M_AVO characteristic.
TT2$M_BLOCK	Block mode capability 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_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$M_DECCRT.
TT2$M_DIALUP	Terminal is a dial-up line. Used by LOGINOUT for the disable dial-up control.

---

[1] 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.

Table 8–6 (Cont.) Extended Terminal Characteristics

Value[1]	Meaning
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 *VAX/VMS DCL Dictionary*.)
TT2$M_DMA	DMA mode. This characteristic enables the use of DMA mode for asynchronous DMA multiplexers. It is ignored by non-DMA controllers.
TT2$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$M_FALLBACK	Output is transformed from the eight-bit multinational character set to a seven-bit ASCII character set on terminals that do not support the eight-bit character set (see Appendix B).
TT2$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.
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).

---

[1] 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.

Table 8–6 (Cont.) Extended Terminal Characteristics

Value[1]	Meaning
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$M_PASTHRU	Terminal is in PASTHRU mode; all input and output data is in seven- or eight-bit binary format (no data interpretation occurs). Data is terminated when the buffer is full or when the data that is read matches the specified terminator. If the characteristic TT$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$M_SYSPWD	System password. This characteristic specifies that the login procedure should require the system password before the user name prompt is displayed.
TT2$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.

[1] 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.

8.3.1 Terminal Characteristics Categories

The set mode and set characteristics functions (see Section 8.4.3) and the SET TERMINAL command are used to change terminal characteristics. The *VAX/VMS DCL Dictionary* describes the SET TERMINAL command.

To customize terminal behavior and usage, the VAX/VMS operating system divides terminal characteristics into six basic categories.

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

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

- 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_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_NOTYPAHEAD
  TT2$M_MODHANGUP
  TT2$M_SECURE
  TT2$M_SETSPEED
  TT2$M_SYSPWD

- User preference—The following characteristics allow the user 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

8.4 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.

8.4.1 Read

When a read function code is issued, the user-specified buffer is filled with characters from the associated terminal. The VAX/VMS operating system provides three 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 occur:

- The user buffer is full.
- The received character is included in a specified terminator mask (see Section 8.4.1.2).

Six device/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 8.4.1.3).

- 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 8–7, IO$M_TIMED)
- P4—the read terminator descriptor block address (see Section 8.4.1.2)
- P5—the starting virtual address of the prompt buffer that is to be written to the terminal; for read with prompt operations using the IO$_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, but no carriage control can be implicitly specified. (Carriage control specifiers are described in Section 8.4.2.2.)

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 the user's 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 not more than 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.)

8.4.1.1 Function Modifier Codes for Read QIO Functions

Eight function modifiers can be specified with IO$_READVBLK, IO$_READLBLK, and IO$_READPROMPT. Table 8–7 lists these function modifiers and IO$_EXTEND, which is described in 8.4.1.3.

Table 8–7 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 8.4.1.3). 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$M_NOECHO, which inhibits all read operation echoing.
IO$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$M_TIMED	The P3 argument specifies the maximum time (seconds) that can elapse between characters received from the terminal; that is, the timeout value for the operation. (Note that because driver timing operates on a one-second timer, a two-second timeout must be specified to guarantee a one-second wait.) The timer starts when the prompt echo is started. If the read time exceeds the time specified in P3, a timeout error (SS$_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. Note that the timer starts when the prompt echo is started.

Table 8–7 (Cont.) Read QIO Function Modifiers for the Terminal Driver

Code	Consequence
	If a read operation is interrupted by either a broadcast write or a synchronous write request, the timer operation is restarted.
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.

8.4.1.2 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 VAX RMS standard terminator set.) The DELETE character (hexidecimal 7F) and eight-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 will terminate 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, 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 8–3. 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 one byte is acceptable. For example, a mask size of 16 bytes allows all seven-bit ASCII characters to be used as terminators; a mask size of 32 bytes allows all eight-bit characters to be used as terminators for eight-bit terminals.

If the terminator mask is all zeros, there are no specified terminators. The read operation ends when the specified number of bytes, that is, 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 8.2.1.2.).

8.4.1.3 Itemlist Read Operations

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

- IO$_READVBLK!IO$M_EXTEND—itemlist read virtual block

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

Figure 8–3 Short and Long Forms of Terminator Mask Quadwords

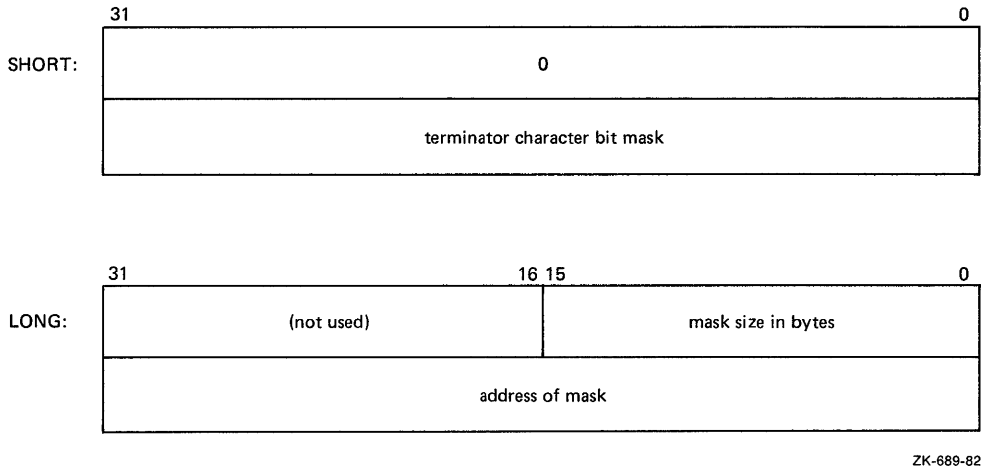
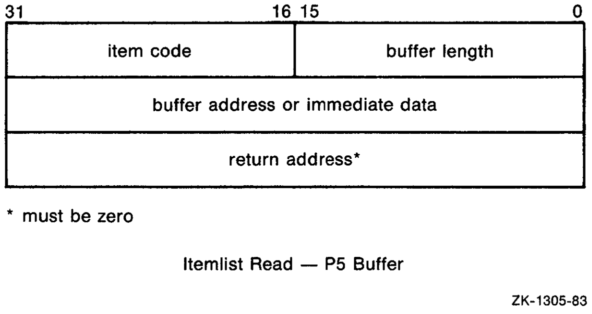

31
0
SHORT:
0
terminator character bit mask
31
16 15
0
LONG:
(not used)
mask size in bytes
address of mask
ZK-689-82

Figure 8–4 Itemlist Read Descriptor


31
16 15
0
item code
buffer length
buffer address or immediate data
return address*
* must be zero
Itemlist Read — P5 Buffer
ZK-1305-83

The itemlist read function code and modifier combination takes the following device/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 8–8.)
- P3—the access mode at which the itemlist is to be probed optional)
- P4—(ignored)
- P5—the address of the itemlist buffer
- P6—the length in bytes of the itemlist buffer

P5 points to a series of item descriptors. Figure 8–4 shows the format for these descriptors. You cannot repeat the same item code in the same item list.

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

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

Item Code	Meaning
TRM$_ALTECHSTR	Alternate echo string. The buffer length word contains the length of the string. The data address word contains the address of the string. The alternate echo string is written to the terminal after the first valid character is entered.
Note:	**Only for character validating read mode editing (TRM$K_EM_RDVERIFY).**
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 8.4.1.4.
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$_PARTESCAPE. This overflow buffer ensures that all the characters in an escape sequence terminator will fit in the user buffer, thus eliminating the need for additional single-character read operations.
TRM$_FILLCHR	A two-byte value that indicates the fill and clear character for TRM$K_EM_RDVERIFY. The first byte of the immediate data longword specifies the clear character; the second byte specifies the fill character.
Note:	**Only for character validating read mode editing (TRM$K_EM_RDVERIFY).**
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.

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

Item Code	Meaning	
TRM$_MODIFIERS	Read modifiers. The immediate data longword contains a 32-bit value that specifies modifiers to read operations, which are defined in $TRMDEF. The buffer length word must be zero. The following bits are defined:	
	TRM$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$M_TM_DSABLMBX	The mailbox is disabled for unsolicited data.
	TRM$M_TM_ESCAPE	A valid ANSI escape sequence is recognized as a valid delimiter for the read operation.
	TRM$M_TM_NOECHO	Characters are not echoed as they are entered at the keyboard.
	TRM$M_TM_NOEDIT	This bit inhibits advanced editing for this read operation.
	TRM$M_TM_NOFILTR	The terminal does not interpret DEL, CTRL/U, or CTRL/R, but passes them to the user. This characteristic explicitly disables line editing.
	TRM$M_TM_NORECALL	This bit inhibits command recall (CTRL/B) by the terminal driver.
	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).

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

Item Code	Meaning	
	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.
	TRM$M_TM_TRMNOECHO	The termination character (if any) is not echoed. There is no formal terminator if the buffer is filled before the terminator is typed.
	Note: All other bits must be zero.	
TRM$_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$K_EM_RDVERIFY. **Note: Only for character validating read mode editing (TRM$K_EM_RDVERIFY).** The format of the character validation string is one 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$_INISTRNG is used. Appendix B lists the multinational character set.	
TRM$_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.	

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

Item Code	Meaning
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. The immediate data longword specifies the maximum time (seconds) for the read operation. TRM$_TIMEOUT automatically sets TRM$M_TM_TIMED. (Note that because driver timing functions on a one-second timer, a two-second timeout must be specified to guarantee a one-second wait.)

8.4.1.4 Read Verify Function

When using the read verify function, the terminal driver performs input validation based on character attributes. 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 will terminate 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 will occur 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, then the read verify is completed with status SS$_OPINCOMPL.

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

8.4.2 Write

Write operations display the contents of a user-specified buffer on the associated terminal. The VAX/VMS operating system provides three basic 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/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.)
- P3 (ignored).
- P4—carriage control specifier except for write physical block operations (Write function carriage control is described in Section 8.4.2.2.)

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 (that is, 8, 16, 24, 32,...).

8.4.2.1 Function Modifier Codes for Write QIO Functions

Five function modifiers can be specified with IO$_WRITEVBLK, IO$_WRITELBLK, and IO$_WRITEPBLK. Table 8–9 lists these function modifiers.

Table 8–9 Write QIO Function Modifiers for the Terminal Driver

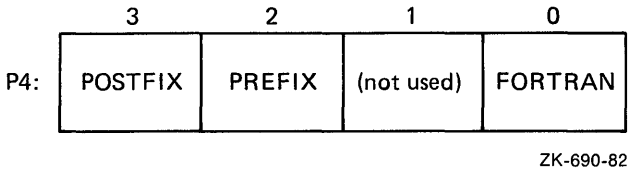
Code	Consequence
IO$M_BREAKTHRU	Allows breakthrough read regardless of the current active state.
IO$M_CANCTRLO	Turns off CTRL/O (if it is in effect) before the write operation. Otherwise, the data may not be displayed.
IO$M_ENABLMBX	Enables use of the mailbox associated with the terminal for notification that unsolicited data is available.
IO$M_NOFORMAT	Allows users 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."

8.4.2.2 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 8–5 shows the P4 longword format.

Figure 8–5 P4 Carriage Control Specifier



Only bytes 0, 2, and 3 in the longword are used. Byte 1 is ignored. If the low-order byte (byte 0) is not 0, the contents of the longword are interpreted as a FORTRAN carriage control specifier. Table 8–10 lists the possible byte 0 values (in hexadecimal) and their meanings.

Table 8–10 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: newline, print buffer contents, return[1])
30	0	Double-space carriage control. (Sequence: newline, newline, print buffer contents, return)
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: newline, print buffer contents)
All other values		Same as ASCII space character: single-space carriage control

[1] A newline 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:

```
Prefix carriage control - Print - Postfix carriage control
```

The prefix and postfix bytes, although interpreted separately, use the same encoding scheme. Table 8–11 shows this encoding scheme in hexadecimal.

With several exceptions, Figure 8–6 shows the prefix and postfix hexadecimal coding that produces the carriage control functions listed in Table 8–10. Prefix and postfix coding provides an alternative way to achieve these controls.

In the first example in Figure 8–6, the prefix/postfix hexadecimal coding for a single-space carriage control (newline, 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 8–11 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, that is, NULL.
0	1–7F	Bits 0 through 6 are a count of newlines (carriage return followed by a line feed).

Bit 7	Bit 6	Bit 5	Bits 0–4	Meaning
1	0	0	1–1F	Output the single ASCII control character specified by the configuration of bits 0 through 4 (seven-bit character set).
1	1	0	1–1F	Output the single ASCII control character specified by the configuration of bits 0 through 4, which are translated as ASCII characters 128 through 159 (eight-bit character set; see Appendix B).

8.4.3 Set Mode

Set mode operations affect the operation and characteristics of the associated terminal line. The VAX/VMS 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. A single function code is provided:

- IO$_SETMODE

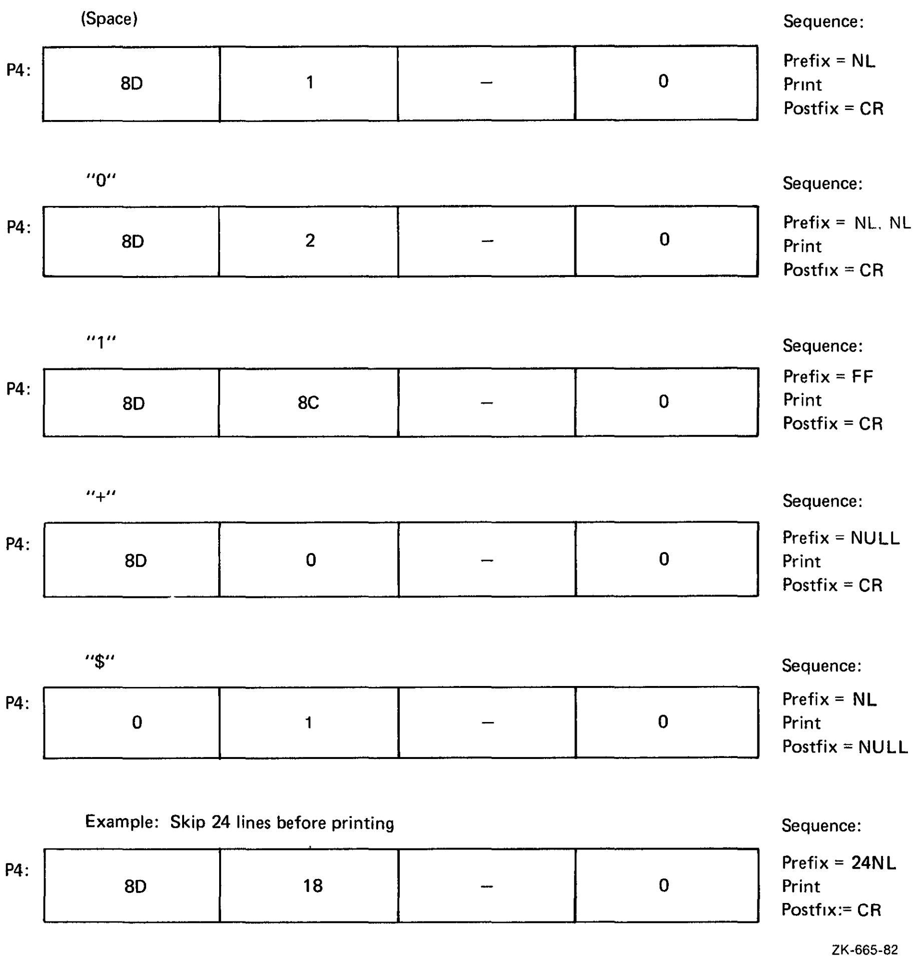
The set characteristics function affects the permanent characteristics of the associated terminal line. Set characteristics is a physical I/O function and requires the privilege necessary to perform physical I/O. A single function code is provided:

- IO$_SETCHAR

The set mode and set characteristics functions take the following device/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)
- P5—parity flags

Figure 8–6 Write Function Carriage Control (Prefix and Postfix Coding)

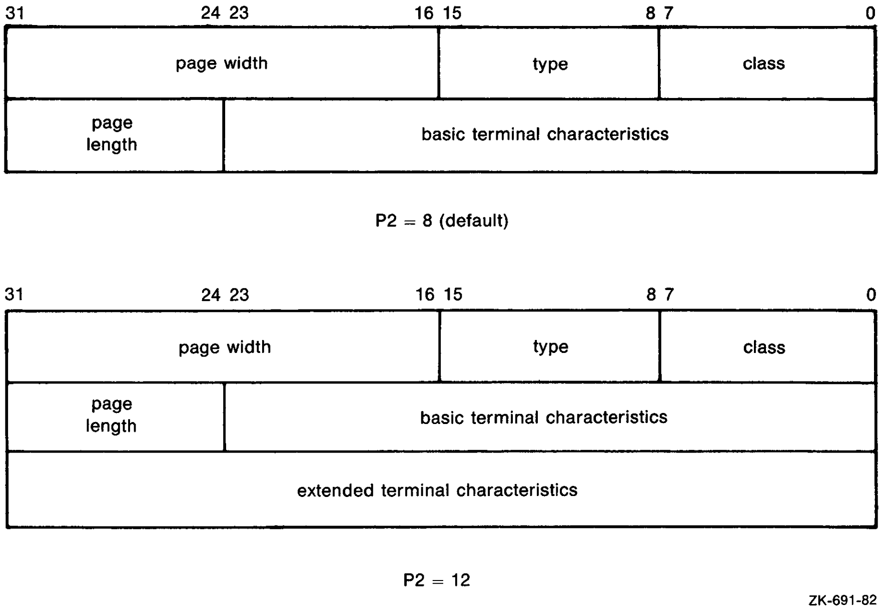

(Space)
P4:
8D
1
–
0
Sequence:
Prefix = NL
Print
Postfix = CR
"0"
P4:
8D
2
–
0
Sequence:
Prefix = NL, NL
Print
Postfix = CR
"1"
P4:
8D
8C
–
0
Sequence:
Prefix = FF
Print
Postfix = CR
"+"
P4:
8D
0
–
0
Sequence:
Prefix = NULL
Print
Postfix = CR
"$"
P4:
0
1
–
0
Sequence:
Prefix = NL
Print
Postfix = NULL
Example: Skip 24 lines before printing
P4:
8D
18
–
0
Sequence:
Prefix = 24NL
Print
Postfix:= CR
ZK-665-82

The P1 argument points to a variable length block, as shown in Figure 8–7. 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 $DCDEF macro, for example, DT$_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 8–5 lists the values for terminal characteristics. Table 8–6 lists the extended terminal characteristics. Characteristics values are defined by the $TTDEF and $TT2DEF macros.

Note: Users should determine that the device is a terminal before performing any set mode function, particularly when using SYS$INPUT or SYS$OUTPUT.

Figure 8–7 Set Mode and Set Characteristics Buffers


31
24 23
16 15
8 7
0
page width
type
class
page length
basic terminal characteristics
P2 = 8 (default)
31
24 23
16 15
8 7
0
page width
type
class
page length
basic terminal characteristics
extended terminal characteristics
P2 = 12
ZK-691-82

The P3 argument defines the device speed, for example, 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 may be specified, not all controllers support all speed combinations. Users should refer to the associated hardware documentation to determine which speeds are supported by their 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 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 8–5.

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, input errors with a parity error flagged are discarded and no error is 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 may result in a character loss.

- TT$M_BREAK—Generate a break if set. The break will be in effect until this bit is turned off.

The following value can be specified in P5 to change the frame size:

- TT$M_ALTFRAME—If set, the low-order four bits of P5 are taken as the frame size. Note that the frame size is for data bits only and is exclusive of parity.

In order to take the existing parity settings, modify them, and use them in the set mode or set characteristic function, the user should 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 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 8.2.5 for information about the SET TERMINAL/FRAME command.

Application programs that change specific 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.

8.4.3.1 Hangup Function Modifier

The hangup function disconnects a terminal that is on a dial-up line. (Dial-up lines are described in Section 8.2.3.) Two 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.

8.4.3.2 Enable CTRL/C AST and Enable CTRL/Y AST Function Modifiers

Both set mode functions can take the enable CTRL/C AST and enable CTRL/Y AST function modifiers. These function modifiers request the terminal driver to queue an AST for the requesting process when the user presses CTRL/C or CTRL/Y. Two 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/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 8–2).

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.

The user 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, that is, on a last-in/first-out basis.

If no enable CTRL/C AST is present, the holder of an enable CTRL/Y AST will receive an AST when CTRL/C is pressed; newline, ^Y, return are echoed.

Figure 8–9 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 the 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 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, CONTINUE, and so on. Thus programs that are run from a command interpreter should not enable CTRL/Y. Also, because ASTs are delivered on a first-in/first-out basis, the command interpreter's AST routine will get 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 8.2.1.2 describes other effects of CTRL/C and CTRL/Y.

8.4.3.3 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. Two combinations of function code and modifier are provided:

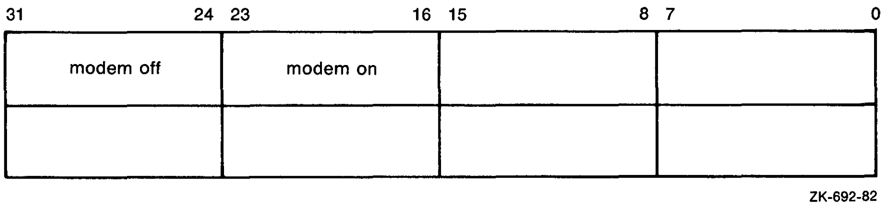
- IO$_SETMODE!IO$M_SET_MODEM!IO$M_MAINT
- IO$_SETCHAR!IO$M_SET_MODEM!IO$M_MAINT

These function code modifier pairs take one device/function-dependent argument:

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

Figure 8–8 shows the format of this block.

Figure 8–8 Set Mode P1 Block



The modem on and modem off fields, in combination or separately, may 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)

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

Note: **Because the DMF32 does not provide the secondary transmitted data signal (Sec Txd), the driver sets the secondary request to send signal instead. To obtain the desired results, users should connect a jumper cable between pins 14 and 19 on the DMF32.**

8.4.3.4 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) will be returned. Both set mode functions can take the loopback function modifier. Two 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 8.2.1.5). Programs that use the loopback function modifier should incorporate a one-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 for 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 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, the user must avoid the possibility of concurrent loopback mode operation on more than one serial line.**

The VAX/VMS operating system provides another function modifier to reset a terminal line previously placed in loopback mode. Two 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 one-second delay to allow the controller to reset the loopback mode after the request is posted.

8.4.3.5 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 the user enters any one of 32 control characters. Two 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/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 8.4.1.2)
- 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:

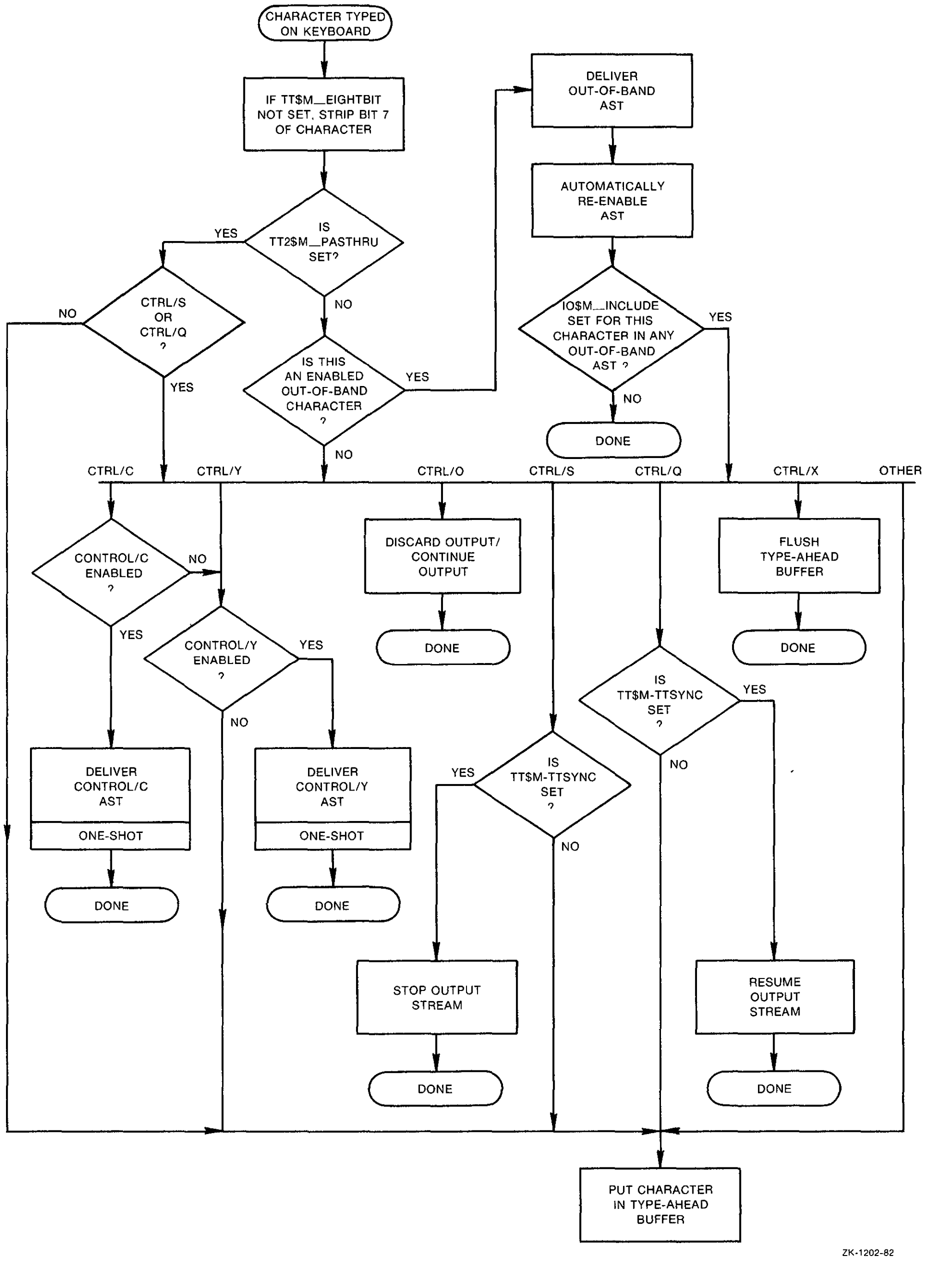
- IO$M_INCLUDE—include the character typed in the data stream
- IO$M_TT_ABORT—allow current read and write operations to be aborted (The IOSB for aborted operations returns the status SS$_CONTROLC.)

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 8–9 shows the relationship of the out-of-band function with some of the control characters.

The user 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.

Figure 8–9 Relationship of Out-of-Band Function with Control Characters


CHARACTER TYPED ON KEYBOARD
IF TT$M_EIGHTBIT NOT SET, STRIP BIT 7 OF CHARACTER
IS TT2$M_PASTHRU SET?
YES
NO
CTRL/S OR CTRL/Q ?
IS THIS AN ENABLED OUT-OF-BAND CHARACTER ?
DELIVER OUT-OF-BAND AST
AUTOMATICALLY RE-ENABLE AST
IO$M_INCLUDE SET FOR THIS CHARACTER IN ANY OUT-OF-BAND AST ?
DONE
CTRL/C
CTRL/Y
CTRL/O
CTRL/S
CTRL/Q
CTRL/X
OTHER
CONTROL/C ENABLED ?
CONTROL/Y ENABLED ?
DISCARD OUTPUT/ CONTINUE OUTPUT
FLUSH TYPE-AHEAD BUFFER
IS TT$M-TTSYNC SET ?
DELIVER CONTROL/C AST
ONE-SHOT
DELIVER CONTROL/Y AST
STOP OUTPUT STREAM
RESUME OUTPUT STREAM
PUT CHARACTER IN TYPE-AHEAD BUFFER
ZK-1202-82

8.4.3.6 Broadcast Function Modifier

The broadcast function modifier allows the user to turn on or turn off selected broadcast requester identifiers (IDs). A single combination of function code and modifier are provided:

- IO$_SETMODE!IO$M_BRDCST

This function code modifier pair takes two device/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 eight bytes)

The first longword of P1 is reserved for use by DIGITAL facilities, as shown in Table 8-12. 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 8–12 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_USERn	Disables broadcasts by images associated with the specified value; *n* can be any decimal integer between 1 and 16

8.4.4 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. Two function codes are provided:

- IO$_SENSEMODE
- IO$_SENSECHAR

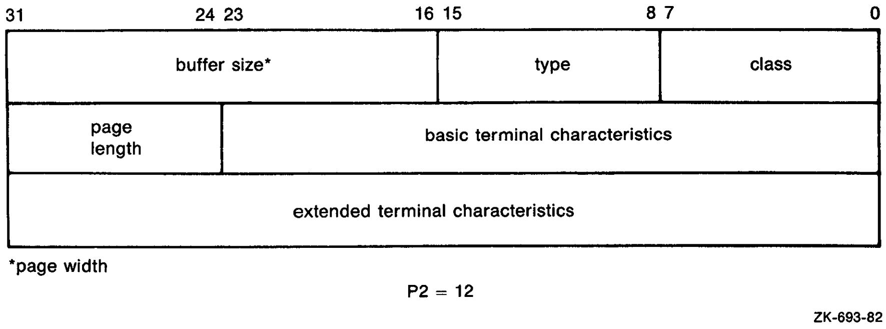
IO$_SENSEMODE returns the temporary characteristics of the terminal, that is, the characteristics associated with the current process, and IO$_SENSECHAR returns the permanent characteristics of the terminal. IO$_SENSEMODE is a logical I/O function and requires no privilege. IO$_SENSECHAR is a physical I/O function and requires the privilege necessary to perform physical I/O.

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

- P1—address of a characteristics buffer
- P2—length of characteristics buffer (default length is 8 bytes)

The P1 argument points to a variable-length block, as shown in Figure 8–10.

Figure 8–10 Sense Mode Characteristics Buffer


31
24 23
16 15
8 7
0
buffer size*
type
class
page length
basic terminal characteristics
extended terminal characteristics
*page width
P2 = 12
ZK-693-82

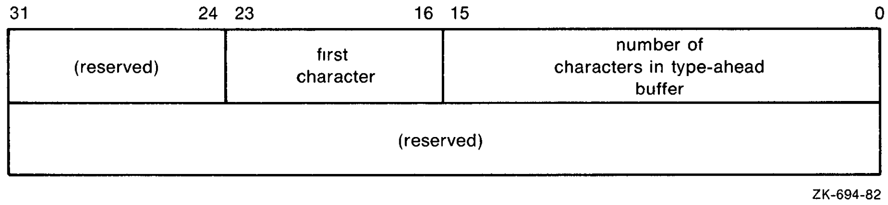
In the buffer, the device class is DC$_TERM, which is defined by the $DCDEF macro. The terminal type is defined by the $DCDEF macro, for example, DT$_LA36. The maximum entry for buffer size (page width) is 255. Table 8–5 lists the values for terminal characteristics. Table 8–6 lists the extended terminal 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 that are described in the next few sections.

8.4.4.1 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 8–11 shows the format of this buffer.

Figure 8–11 Sense Mode Characteristics Buffer (type-ahead)


31
24 23
16 15
0
(reserved)
first character
number of characters in type-ahead buffer
(reserved)
ZK-694-82

8.4.4.2 Read Modem Function Modifier

The read modem function modifier allows access to controller-dependent information. Two combinations of function code and modifier are provided:

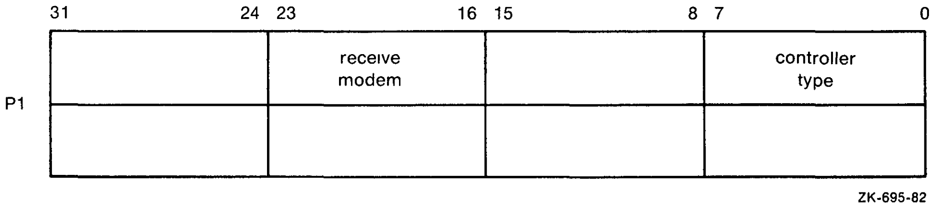
- IO$_SENSEMODE!IO$M_RD_MODEM
- IO$_SENSECHAR!IO$M_RD_MODEM

These function code modifier pairs take one device/function-dependent argument:

- P1—the address of a quadword block

Figure 8–12 shows the format of this block.

Figure 8–12 Sense Mode P1 Block



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 different 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

8.4.4.3 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 8–12 in Section 8.4.3.6). A single combination of function code and modifier are provided:

- IO$_SENSEMODE!IO$M_BRDCST

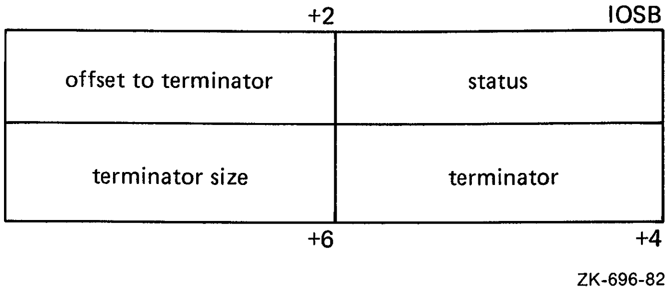
This function code modifier pair takes two device/function-dependent arguments:

- 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

8.5 I/O Status Block

The I/O status block (IOSB) formats for the read, write, set mode, set characteristics, sense mode, and sense characteristics I/O functions are shown in Figures 8–13, 8–15, and 8–16. Figure 8–14 shows the IOSB format for the itemlist read function. Appendix A lists the status returns for these functions. (The *VAX/VMS System Messages and Recovery Procedures Reference Manual* provides explanations and suggested user actions for these returns.)

Figure 8–13 IOSB Contents — Read Function



In Figure 8–13, the offset to terminator at IOSB+2 is the count of characters before the terminator character (see Section 8.4.1.2). The terminator character(s) is in the buffer at the offset specified in IOSB+2. When the buffer is full, the offset 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 8.2.1.4). The sum of IOSB+2 and IOSB+6 is the number of characters in the buffer.

In Figure 8–14 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 8–14 IOSB Contents — Itemlist Read Function

<table>
<tr><td colspan="2">offset to terminator</td><td colspan="2">status</td></tr>
<tr><td>cursor position from EOL</td><td>terminator length</td><td>(reserved)</td><td>terminator character</td></tr>
</table>

IOSB Contents: Itemlist Read Function

ZK-1306-83

For normal read operations, IOSB+5 and IOSB+7 are returned as zero. For itemlist read operations, IOSB+5 is returned as –1, and IOSB+7 is the cursor offset from the end of line when the read operation terminates.

In Figure 8–15 note that the remote terminal driver does not return the number of lines output or the cursor position.

Figure 8–15 IOSB Contents — Write Function

byte count	status
0	0

IOSB Contents: Write Funtion

ZK-1307-83

Figure 8–16 IOSB Contents — Set Mode, Set Characteristics, Sense Mode, and Sense Characteristics Functions

<table>
<tr><td>receive speed*</td><td>transmit speed</td><td colspan="2">status</td></tr>
<tr><td>0</td><td>parity flags</td><td>LF fill count</td><td>CR fill count</td></tr>
</table>

*only specified if different than transmit speed

ZK-698-82

8.6 Programming Examples

This section contains two programming examples: one to show several I/O operations using the full-duplex capabilities of the terminal, and the other to show a typical read verify operation.

Example 8–1 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 the user enters a message on the terminal, the read AST routine prints an acknowledgment message and queues another read request. If the user presses CTRL/C, the associated AST routine cancels the I/O operation on the assigned channel and exits to the command interpreter.

The second program (Example 8–2) follows Example 8–1.

Example 8–1 Terminal Program Example

```
        .TITLE  FULL_DUPLEX TERMINAL PROGRAMMING EXAMPLE
        .IDENT  /04/
; ******************************************************************
;
;                         TERMINAL PROGRAM
;
; ******************************************************************
        .SBTTL  DECLARATIONS

;
; DEFINE SYMBOLS
;
        $IODEF                  ; DEFINE I/O FUNCTION CODES
        $QIODEF                 ; DEFINE QIO DEFINITION CODES
        $TRMDEF                 ; DEFINE ITEMLIST READ CODES

;
; 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
;
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
```

(Continued on next page)

Example 8–1 (Cont.) Terminal Program Example

```
;
; 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=^XOD                 ; CARRIAGE RETURN
        LF=^XOA                 ; 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
;
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:
        $QIO    FUNC=IO$_WRITEVBLK!IO$M_BREAKTHRU!IO$M_REFRESH,-
                EFN=1           ; QIO PACKET
```

(Continued on next page)

Example 8–1 (Cont.) Terminal Program Example

```
;
; TIMER STORAGE
;
WAITIME:
        .LONG   -10*1000*1000*3,-1      ; 3 SECOND DELTA TIME
TIME:
        .BLKQ   1               ; CURRENT STORAGE TIME USED FOR
                                ; RANDOM NUMBER

        .SBTTL  START - MAIN ROUTINE
;++
;
; FUNCTIONAL DESCRIPTION:
;
; *******************************************************************
;
;                       START PROGRAM
;
; *******************************************************************
;
;       The following code performs initialization functions.
;       The program assumes 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
;
;--
START:
        .WORD                           ; ENTRY MASK
        $ASSIGN_S       DEVNAM=TT_DESC,-; ASSIGN TERMINAL CHANNEL USING
                        CHAN=TT_CHAN    ; LOGICAL NAME AND CHANNEL NUMBER
        BLBS    R0,10$                  ; NO ERROR IF SET
        BRW     ERROR                   ; ERROR
10$:    BSBW    CHANGE_CHARACTERISTICS  ; CHANGE THE CHARACTERISTICS OF
                                        ; TERMINAL
        BSBW    ENABLE_CTRLCAST         ; ALLOW CONTROL/C TRAPS
        BSBW    ENABLE_OUTBANDAST       ; ENABLE CONTROL A OUT-OF-BAND AST
        BSBW    ENABLE_READ             ; QUEUE READ
        MOVZWL  TT_CHAN,WRITE_QIO+8     ; INSERT CHANNEL INTO
                                        ; STATIC QIO PACKET
;
; THIS LOOP OUTPUTS A MESSAGE BASED ON A RANDOM NUMBER AND THEN
; DELAYS FOR 3 SECONDS
;
LOOP:
        $GETTIM_S       TIMADR=TIME      ; GET RANDOM TIME
        BLBS    R0,10$                   ; NO ERROR IF SET
        BRW     ERROR
10$:    EXTZV   #6,#2,TIME,R0            ; LOAD RANDOM BITS INTO
                                         ; SWITCH
        MOVQ    ARRAY[R0],-              ; LOAD MESSAGE ADDRESS
                WRITE_QIO+QIO$_P1        ; AND SIZE INTO QIO
                                         ; PACKET
```

(Continued on next page)

Example 8–1 (Cont.) Terminal Program Example

```
;
; ISSUE QIO WRITE USING PACKET DEFINED IN DATA AREA
;
        $QIOW_G  WRITE_QIO
        BLBS     RO,5$                   ; NO ERROR IF SET
        BRW      ERROR
;
; DELAY FOR 3 SECONDS BEFORE ISSUING NEXT MESSAGE
;
5$:
        $SETIMR_S  EFN=#2,DAYTIM=WAITIME        ; TIMER SERVICE
                                         ; WILL SET EVENT FLAG
                                         ; IN 3 SECONDS
        BLBS    RO,20$                   ; NO ERROR IF SET
        BRW     ERROR
20$:    $WAITFR_S      EFN=#2            ; WAIT FOR EVENT FLAG
        BLBS    RO,LOOP                  ; NO ERROR IF SET
        BRW     ERROR
        .SBTTL  CHANGE_CHARACTERISTICS - CHANGE CHARACTERISTICS OF
                                        TERMINAL
;++
;
; FUNCTIONAL DESCRIPTION:
;
;       Routine to change the characteristics of the terminal.
;
; INPUT PARAMETERS:
;       NONE
;
; OUTPUT PARAMETERS:
;       RO - status from $QIO call.
;       R1 - R5 destroyed
;
;--
;
CHANGE_CHARACTERISTICS:
        $QIO_S  CHAN=TT_CHAN,-          ; GET CURRENT TERMINAL
                FUNC=#IO$_SENSEMODE,-   ; CHARACTERISTICS
                P1=OLDCHAR_BUF,-
                P2=#OLDCHAR_BUF_LEN
        BLBC    RO,10$                  ; ERROR IF CLEAR
        $DCLEXH_S EXIT_HANDLER_BLOCK    ; DECLARE EXIT HANDLER TO RESET
                                        ; CHARACTERISTICS
        BLBS    RO,5$
        BRW     ERROR
5$:     MOVC3   #OLDCHAR_BUF_LEN,-      ; MOVE OLD CHARACTERISTICS INTO
                OLDCHAR_BUF,-           ; NEW CHARACTERISTICS BUFFER
                NEWCHAR_BUF
        BISL2   #TT$M_NOBRDCST,-        ; SET NOBROADCAST BIT
                NEWCHAR_BUF+4           ; ...
        $QIO_S  CHAN=TT_CHAN,-          ; SET CURRENT TERMINAL
                FUNC=#IO$_SETMODE,-     ; CHARACTERISTICS
                P1=NEWCHAR_BUF,-
                P2=#NEWCHAR_BUF_LEN
        BLBS    RO,20$                  ; NO ERROR IF SET
10$:    BRW     ERROR
20$:    RSB
```

(Continued on next page)

Example 8–1 (Cont.) Terminal Program Example

```
        .SBTTL  ENABLE_CTRLCAST - ENABLE CONTROL C AST
;++
;
; FUNCTIONAL DESCRIPTION:
;
;       Routine to allow CONTROL C recognition.
;
; INPUT PARAMETERS:
;       NONE
;
; OUTPUT PARAMETERS:
;       NONE
;
;--
;
ENABLE_CTRLCAST:
        $QIOW_S CHAN=TT_CHAN,-
                FUNC=#IO$_SETMODE!IO$M_CTRLCAST,-
                P1=CTRLCAST,-           ; AST ROUTINE ADDRESS
                P3=#3                   ; USER MODE
        BLBS    RO,10$                  ; NO ERROR IF SET
        BRW     ERROR
10$:    RSB
        .SBTTL  ENABLE_OUTBANDAST - ENABLE CONTROL A AST
;++
;
; FUNCTIONAL DESCRIPTION:
;
;       Routine to allow CONTROL A recognition.
;
; INPUT PARAMETERS:
;       NONE
;
; OUTPUT PARAMETERS:
;       NONE
;
ENABLE_OUTBANDAST:
        $QIOW_S CHAN=TT_CHAN,-
                FUNC=#IO$_SETMODE!IO$M_OUTBAND,-
                P1=CTRLAAST,-           ; AST ROUTINE ADDRESS
                P2=#CNTRLA_MASK,-       ; CHARACTER MASK
                P3=#3                   ; USER MODE
        BLBS    RO,10$                  ; NO ERROR IF SET
        BRW     ERROR
10$:    RSB
        .SBTTL  ENABLE_READ - QUEUE A READ TO THE TERMINAL.
;++
;
; FUNCTIONAL DESCRIPTION:
;
;       Routine to queue a read to the terminal.
;
; INPUT PARAMETERS:
;       NONE
;
; OUTPUT PARAMETERS:
;       NONE
;
; DEFINE ITEM LIST FOR ITEMLIST READ
ITEM_LST:
        ITEM    0,MODIFIERS,-                   ; CONVERT LOWER CASE TO
                TRM$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 8–1 (Cont.) Terminal Program Example

```
ITEM_LEN=.-ITEM_LST
MASK_ADDR:
        .LONG   1@^XD                   ; TERMINATOR MASK IS <CR>
        .WORD   1@4                     ; AND "$"
ENABLE_READ:
        $QIO_S  CHAN=TT_CHAN,-          ; MUST NOT BE QIOW FORM
                        -               ; OR READ WILL BLOCK
                        -               ; PROCESS
                FUNC=#IO$_READVBLK!IO$M_EXTEND,-
                IOSB=IN_IOSB,-
                ASTADR=READAST,-        ; AST ROUTINE TO EXECUTE
                P1=IN_BUF,-             ; ON
                P2=#IN_BUFLEN,-
                P5=#ITEM_LST,-          ; ITEMLIST READ ADDRESS
                P6=#ITEM_LEN            ; ITEMLIST READ SIZE
        BLBS    R0,10$                  ; NO ERROR IF SET
        BRW     ERROR
; THE QUEUED READ WILL NOT AFFECT WRITES DUE TO THE FACT THAT BREAKTHRU
; HAS BEEN SET FOR THE WRITES.
10$:    RSB
        .SBTTL  READAST - AST ROUTINE FOR READ COMPLETION
;++
;
; FUNCTIONAL DESCRIPTION:
;
;       AST routine to execute on read completion.
;
; INPUT PARAMETERS:
;       NONE
;
; OUTPUT PARAMETERS:
;       NONE
;
;--
;
READAST:
        .WORD   ^M<R2,R3,R4,R5>       ; PROCEDURE ENTRY MASK
        BLBS    IN_IOSB,10$             ; CHECK IOSB FOR SUCCESS
        MOVZWL  IN_IOSB,R0              ; PUT ERROR STATUS IN R0
        BRW     ERROR                   ; EXIT WITH ERROR
10$:    MOVZWL  IN_IOSB+2,R0            ; GET # CHARACTERS READ INTO R0
        ADDL2   #ACK_MSGLEN,R0          ; ADD SIZE OF FIXED ACKNOWLEDGE
                                        ; MESSAGE
        $QIO_S  CHAN=TT_CHAN,-          ; ISSUE ACKNOWLEDGE
        FUNC=#IO$_WRITEVBLK,-           ; MESSAGE
        P1=ACK_MSG,-
        P2=R0
;
; PROCESS READ MESSAGE
;
;        .
;        .
;        .
;(user-provided code to decode command inserted here)
;        .
;        .
;        .
        BSBW     ENABLE_READ            ; QUEUE NEXT READ
        RET                             ; RETURN TO MAINLINE LOOP

        .SBTTL  CTRLAAST - AST ROUTINE FOR CONTROL A
        .SBTTL  CTRLCAST - AST ROUTINE FOR CONTROL C
        .SBTTL  ERROR - EXIT ROUTINE
```

(Continued on next page)

Example 8–1 (Cont.) Terminal Program Example

```
;++
;
; FUNCTIONAL DESCRIPTION:
;
;       AST routine to execute when CONTROL C or CONTROL A is entered.
;
; INPUT PARAMETERS:
;       NONE
;
; OUTPUT PARAMETERS:
;       NONE
;
CTRLCAST:
CTRLAAST:
        .WORD   ^M<>                    ; PROCEDURE ENTRY MASK
        MOVL    #1,R0                     ; PUT SUCCESS IN R0
ERROR:  $EXIT_S R0                        ; EXIT
        RSB
        .SBTTL  EXIT_HANDLER - EXIT HANDLER ROUTINE
;++
;
; FUNCTIONAL DESCRIPTION:
;
;       Exit handler routine to execute when image exits.  It will cancel
;       and outstanding I/O on this channel, and reset the terminal
;       characteristics to their original state.
;
; INPUT PARAMETERS:
;       NONE
;
; OUTPUT PARAMETERS:
;       NONE
;
;--
;
EXIT_HANDLER:
        .WORD                             ; PROCEDURE ENTRY MASK
        $CANCEL_S       CHAN=TT_CHAN      ; FLUSH ANY I/O ON QUEUE
        $QIO_S  CHAN=TT_CHAN,-            ; RESET TERMINAL CHARACTERISTICS
                FUNC=#IO$_SETMODE,-
                P1=OLDCHAR_BUF,-
                P2=#OLDCHAR_BUF_LEN
        RET
        .END     START
```

Example 8–2 is an example of the read verify function. The program shows a typical build of item lists (both the right and left fields), channel assignment, a right- and left-justified read verify operation, and then the read QIO operation.

Example 8–2 Read Verify Program Example

```
        .TITLE READ_VERIFY - Read Verify Coding Example
        .IDENT  'V04-000'
        .SBTTL  DECLARATIONS
;
; INCLUDE FILES:
;
        $TRMDEF
;
; MACROS:
;
.MACRO ITEM LEN=0,CODE,VALUE
        .WORD   LEN
        .WORD   TRM$_'CODE'
        .LONG   VALUE
        .LONG   0
.ENDM ITEM
;
; EQUATED SYMBOLS:
;
INBUF_LEN = 20
ESC = ^X1B
;
; 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
```

(Continued on next page)

Example 8–2 (Cont.) Read Verify Program Example

```
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
        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/
```

(Continued on next page)

Example 8–2 (Cont.) Read Verify Program Example

```
        .ENTRY  READ_VERIFY,^M<>
        ;
        ; Assign channel to SYS$INPUT
        ;
        $ASSIGN_S -
                CHAN = TT_CHAN -
                DEVNAM = SYSINPUT                  ; SYS$INPUT
        BLBC    RO,ERROR                           ; BRANCH ON ERROR
        ;
        ; Clear the screen
        ;
        CLRQ    -(SP)
        CALLS   #2,G^SCR$ERASE_PAGE
        BLBC    RO,ERROR
        ;
        ; Do the right justified read
        ;
        PUSHL   #R_ITEM_LIST_LEN
        PUSHAB  R_ITEM_LIST
        CALLS   #2,DO_READ
        BLBC    RO,ERROR
        ;
        ; Do the left justified read
        ;
        PUSHL   #L_ITEM_LIST_LEN
        PUSHAB  L_ITEM_LIST
        CALLS   #2,DO_READ
        BLBC    RO,ERROR
ERROR:
        RET
;++
;
; DO_READ - do actual QIO
;
; INPUTS:
;
;       4(AP)   - address of itemlist
;       8(AP)   - length of itemlist
;
;--
.ENTRY  DO_READ, ^M<>
        $QIOW_S CHAN = TT_CHAN -
                FUNC = #<IO$_READVBLK!IO$M_EXTEND>,-
                IOSB = IN_IOSB,-
                p1 = inbuf,-
                p2 = #inbuf_len,-
                p5 = 4(AP),-
                P6 = 8(AP)
        BLBC    RO,10$                     ; BRANCH ON ERROR
        ; .
        ; .  handle the input...
        ; .
10$:    RET
        .END READ_VERIFY
```

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

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$_IDXFILEFULL
SS$_ILLCNTRFUNC	SS$_NODISKQUOTA	SS$_NOMOREFILES
SS$_NOPRIV	SS$_NOQFILE	SS$_NOSUCHFILE

[1] The second longword of the IOSB contains a job controller status code.

QIO Status Returns		
SS$_NOTAPEOP	SS$_NOTLABELMT	SS$_NOTPRINTED[1]
SS$_NOTVOLSET	SS$_OVRDSKQUOTA	SS$_QFACTIVE
SS$_QFNOTACT	SS$_SUPERSEDE	SS$_TAPEPOSLOST
SS$_TOOMANYVER	SS$_WRITLCK	SS$_WRONGACP

[1] The second longword of the IOSB contains a job controller status code.

A.2 Card Reader Driver

Functions	Arguments	Modifiers
IO$_READLBLK IO$_READVBLK IO$_READPBLK	P1 - buffer address P2 - byte count	IO$M_BINARY IO$M_PACKED
IO$_SETMODE IO$_SETCHAR	P1 - characteristics buffer address	(none)
IO$_SENSEMODE	(none)	(none)

QIO Status Returns			
SS$_ABORT	SS$_DATAOVERUN	SS$_ENDOFFILE	SS$_NORMAL

A.3 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)

---

[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$_SEARCH	P1 - read/write head position	(none)
IO$_SEEK[4]	P1 - seek to specified cylinder	(none)
IO$_FORMAT	P1 - RX02 density	(none)
IO$_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]

[4] Not for DSA disks

[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.4 Laboratory Peripheral Accelerator Driver

Functions	Arguments	Modifiers
IO$_LOADMCODE	P1 - starting address of microcode to be loaded P2 - load byte count P3 - starting microprogram address to receive microcode	(none)
IO$_STARTMPROC	(none)	(none)

I/O Function Codes

Functions	Arguments	Modifiers
IO$_INITIALIZE	P1 - address of initialize command table P2 - initialize command buffer length	(none)
IO$_SETCLOCK	P2 - mode of operation P3 - clock control and status P4 - real-time clock preset value (two's complement)	(none)
IO$_STARTDATA	P1 - data transfer command table address P2 - data transfer command table length P3 - normal completion AST address P4 - overrun completion AST address	IO$_SETEVF

High-Level Language Subroutines	Functions
LPA$ADSWP	Start A/D converter sweep.
LPA$DASWP	Start D/A converter sweep.
LPA$DISWP	Start digital input sweep.
LPA$DOSWP	Start digital output sweep.
LPA$LAMSKS	Specify LPA11-K controller and digital mask words.
LPA$SETADC	Specify channel select parameters.
LPA$SETIBF	Specify buffer parameters.
LPA$STPSWP	Stop sweep.
LPA$CLOCKA	Set Clock A rate.
LPA$CLOCKB	Set Clock B rate.
LPA$XRATE	Compute clock rate and preset value.
LPA$IBFSTS	Return buffer status.
LPA$IGTBUF	Return next available buffer.
LPA$INXTBF	Alter buffer order.
LPA$IWTBUF	Return next buffer or wait.
LPA$RLSBUF	Release buffer to LPA11-K.
LPA$RMVBUF	Remove buffer from device queue.
LPA$CVADF	Convert A/D input to floating point.
LPA$FLT16	Convert unsigned integer to floating point.
LPA$LOADMC	Load microcode and initialize LPA11-K.

QIO Status Returns		
SS$_ABORT	SS$_BADPARAM	SS$_BUFNOTALIGN
SS$_CANCEL	SS$_CTRLERR	SS$_DATACHECK
SS$_DEVACTIVE	SS$_DEVCMDERR	SS$_DEVREQERR
SS$_EXQUOTA	SS$_INSFBUFDP	SS$_INSFMAPREQ
SS$_INSFMEM	SS$_IVBUFLEN	SS$_IVMODE
SS$_MCNOTVALID	SS$_PARITY	SS$_POWERFAIL
SS$_TIMEOUT		

A.5 Line Printer Driver

Functions	Arguments	Modifiers
IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1 - buffer address P2 - buffer size P3 - (ignored) P4 - carriage control specifier[1]	(none)
IO$_SENSEMODE	(none)	(none)
IO$_SETMODE IO$_SETCHAR	P1 - characteristics buffer address	(none)

[1] Only for IO$_WRITEVBLK and IO$_WRITELBLK

QIO Status Returns			
SS$_ABORT	SS$_ACCVIO	SS$_CANCEL	SS$_NORMAL

A.6 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]

[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

Functions	Arguments	Modifiers
IO$_SETMODE IO$_SETCHAR	P1 - characteristics buffer address P2 - characteristics buffer length[9]	
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$M_NOWAIT
IO$_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$_PARITY	SS$_TIMEOUT
SS$_UNSAFE	SS$_VOLINV	SS$_WRITLCK

A.7 Mailbox Driver

Functions	Arguments	Modifiers
IO$_READVBLK IO$_READLBLK IO$_READPBLK IO$_WRITEVBLK IO$_WRITELBLK IO$_WRITEPBLK	P1 - buffer address P2 - buffer size	IO$M_NOW
IO$_WRITEOF	(none)	IO$M_NOW
IO$_SETMODE!IO$M_READATTN IO$_SETMODE!IO$M_WRTATTN	P1 - AST address P2 - AST parameter P3 - access mode	(none)
IO$_SETMODE!IO$M_SETPROT	P2 - volume protection mask	(none)

QIO Status Returns			
SS$_ABORT	SS$_BUFFEROVF	SS$_ENDOFFILE	SS$_NORMAL

A.8 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]

---

[1] Only for IO$_READPROMPT

[2] Do not specify with other modifiers. (Itemlist read function.)

Functions	Arguments	Modifiers
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]
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$M_MAINT
IO$_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

---

[3] Only for IO$_WRITELBLK and IO$_WRITEVBLK

[4] Only with IO$M_OUTBAND

[5] Only with IO$M_MAINT

QIO Status Returns

- SS$_ABORT
- SS$_BADESCAPE
- SS$_BADPARAM
- SS$_CANCEL
- SS$_CONTROLC
- SS$_CONTROLO
- SS$_CONTROLY
- SS$_DATAOVERUN
- SS$_INCOMPAT
- SS$_NORMAL
- SS$_PARITY
- SS$_PARTESCAPE
- SS$_TIMEOUT

B Tables

B.1 DEC Multinational Character Set

Table B–1 lists the DEC Multinational Character Set. The DEC Multinational Character set is an eight-bit character set with 256 characters. The first 128 characters in the set correspond to the ASCII character set. Characters with numbers greater than 127 can only be entered on VT52- and VT100-type terminals from screen mode. The *EDT Editor 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)
13	023	019	DC3	device control 3 (CTRL/S)
14	024	020	DC4	device control 4 (CTRL/T)

---

[1] The ALTMODE and DELETE characters (decimal 125, 126, and 127) are also control characters.

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
ASCII Control Characters[1]				
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
34	064	052	4	four
35	065	053	5	five

---

[1] The ALTMODE and DELETE characters (decimal 125, 126, and 127) are also control characters.

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Special and Numeric Characters		
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 Alpha Characters		
40	100	064	@	commercial at
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
56	126	086	V	uppercase V
57	127	087	W	uppercase W
58	130	088	X	uppercase X

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alpha Characters		
59	131	089	Y	uppercase Y
5A	132	090	Z	uppercase Z
5B	133	091	[	opening bracket
5C	134	092	\	backslash
5D	135	093	]	closing bracket
5E	136	094	^	circumflex
5F	137	095	_	underline (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
7A	172	122	z	lowercase z
7B	173	123	{	opening brace
7C	174	124	|	vertical line
7D	175	125	}	closing brace (ALTMODE)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alpha Characters		
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
9E	236	158	PM	privacy message
9F	237	159	APC	application
A0	240	160	—	[reserved]
A1	241	161	¡	inverted exclamation mark

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alpha Characters		
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
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)

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alpha Characters		
C5	305	197	Å	uppercase A with ring
C6	306	198	Æ	uppercase AE with 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
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

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alpha Characters		
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)
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

Table B–1 (Cont.) DEC Multinational Character Set

Hex Code	Octal Code	Decimal Code	Char or Abbrev.	Description
		ASCII Alpha Characters		
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 8). 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- or VT200-family user's guide). Unless otherwise noted, the operation of escape sequences and modes is identical to the particular VT100- or VT200-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			
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			

---

[1] Terminal characteristics. Prefix is TT2$M_

Table B-2 (Cont.) Sequences and Modes

Name	Valid Parameters	ANSICRT	DECCRT	AVO	EDIT	BLOCK[1]
		ANSI-Defined Escape Sequences				
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
		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		

---

[1] Terminal characteristics. Prefix is TT2$M_

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)						
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

Index

A

ACP control function • 1-34
  disk quotas • 1-36
  magnetic tape positioning • 1-35
  miscellaneous disk • 1-36
  quota file transfer block • 1-37
ACP function • 1-2
  arguments • 1-2
  attributes • 1-17 to 1-20
  IO$_ACCESS • 1-9, 1-11, 1-16, 1-28
  IO$_ACPCONTROL • 1-9, 1-34
  IO$_CREATE • 1-11, 1-12, 1-16, 1-25
  IO$_DEACCESS • 1-15, 1-16, 1-31
  IO$_DELETE • 1-9, 1-33
  IO$_MODIFY • 1-9, 1-12, 1-15, 1-16, 1-31
  IO$_MOUNT • 1-33
  major • 1-24
ACP-QIO interface
  access file function • 1-28
  access subfunction • 1-11
  ACP control function • 1-34
  ANSI standard • 1-2, 1-36
  arguments • 1-2
    disk quota • 1-37
  attribute control block • 1-16
  attributes • 1-17 to 1-20
  attributes statistics block • 1-23
  create file function • 1-25
    disk • 1-27
    magnetic tape • 1-28
  deaccess file function • 1-31
  delete file function • 1-33
  description • 1-1
  FIB (file information block) • 1-3
    See also FIB (file information block)
  file characteristics • 1-21
  function codes • A-1
  function modifiers • 1-2
    IO$M_ACCESS • 1-11, 1-25, 1-28
    IO$M_CREATE • 1-25, 1-27, 1-28
    IO$M_DELETE • 1-25, 1-27, 1-33
    IO$M_DMOUNT • 1-34, 1-36
  I/O operations • 1-1
  I/O status block • 1-39
  record attributes area • 1-21
    values • 1-21
ACP-QIO interface (cont'd.)
  serious exception (EOT) • 1-26, 1-29, 1-35, 1-36
  status returns • A-1
  VAX BLISS-32 programming • 1-2
  VAX MACRO programming • 1-1
  XQP (extended QIO processor) • 1-1
ACP subfunction • 1-8
  access • 1-11
  directory lookup • 1-9
  extend • 1-12, 1-39
  read/write attributes • 1-16
  truncate • 1-15
ALTMODE key • 8-20
ANSI escape sequence • B-9
Argument
  device/function-dependent • 1-2
  list • A-1 to A-8
  LPA11-K subroutine • 4-14
ASCII (8-bit) code • 2-7
ASCII character set
  See DEC Multinational Character Set
AST (asynchronous system trap)
  quota • 3-15, 4-12, 6-8, 7-6, 8-41
Attention AST
  read mailbox • 7-8
  terminal • 8-41
  write mailbox • 7-9

B

Baud rate
  terminal • 8-39
BOT (beginning-of-tape)
  See Magnetic tape, BOT marker
Broadcast message • 8-16, 8-20, 8-22, 8-45
Buffered I/O quota • 3-15, 6-8, 7-6
Buffer overrun
  LPA11-K • 4-10

C

Card reader
  capabilities • 2-1
  card punch combinations • 2-1

Card reader (cont'd.)
  026 card reader code • 2-2, 2-7
  029 card reader code • 2-2, 2-7
  code • 2-7
  device characteristics • 2-3
  driver • 2-1
  end-of-file status • 2-2
  error recovery • 2-2
  failure categories • 2-2
  function codes • 2-4, A-2
  function modifiers
    IO$M_BINARY • 2-1, 2-5
    IO$M_PACKED • 2-1, 2-5
  I/O functions
    IO$_READLBLK • 2-4
    IO$_READPBLK • 2-4
    IO$_READVBLK • 2-4
    IO$_SENSEMODE • 2-6
    IO$_SETCHAR • 2-9
    IO$_SETMODE • 2-6
  I/O status block • 2-9
  read function • 2-4
  read modes • 2-1
  sense mode function • 2-6
  set mode function • 2-6
  set translation mode • 2-2
  status returns • A-2
  supported device • 2-1
  SYS$GETDVI • 2-3
Carriage control
  line printer • 5-6
  terminal • 8-35
Character
  formatting on line printer • 5-2
  terminal terminator • 8-28
Character set
  multinational • B-1
  terminal lowercase • 8-20
Clock rate
  LPA11-K • 4-8
CONNECT command • 8-16
Console disk
  See RX01 console disk
Console terminal • 8-1
Control character
  list • B-1
  terminal • 8-3 to 8-6, 8-9
Control sequence
  terminal • 8-8
Create file function • 1-25
CTRL/x
  See Terminal, control characters

D

Data buffer
  LPA11-K • 4-11
Data check
  disk • 3-9, 3-21
  magnetic tape • 6-4, 6-12, 6-14
Data security erase
  magnetic tape • 6-21
Data transfer command table
  LPA11-K • 4-10
Data transfer start command
  LPA11-K • 4-10
Data transfer stop command
  LPA11-K • 4-12
Data underrun/overrun
  LPA11-K • 4-10
Deaccess file function • 1-31
DEC026 card reader code • 2-2, 2-7
DEC029 card reader code • 2-2, 2-7
DEC Multinational Character Set • B-1
Delete file function • 1-33
DELETE key • 8-4
Device characteristics
  card reader • 2-3
  disk • 3-14
  line printer • 5-3
  LPA11-K device • 4-3
  magnetic tape • 6-6
  mailbox • 7-5
  terminal • 8-18
DHU11 device • 8-1
DHV11 device • 8-1
Dial-up line • 8-11
DIGITAL-private escape sequence • B-9
Direct I/O quota • 3-15, 6-8
Directory lookup subfunction • 1-9
DISCONNECT command • 8-16
Disk
  See also DSA disk
  ACP control function • 1-36
  ACP operation
    creating file • 1-27
    deaccessing file • 1-31
  available function • 3-24
  Backup Utility • 3-13
  capabilities • 3-6
  data check • 3-9, 3-21
  device characteristics • 3-14
  driver • 3-1
  dual porting • 3-7

Disk
  dual porting (cont'd.)
    DSA disks • 3-8
    restrictions • 3-8
  error recovery • 3-10
  file attributes • 3-9
  function codes • 3-15, A-2
  function modifiers
    IO$M_DATACHECK • 3-9, 3-21
    IO$M_DELDATA • 3-22
    IO$M_ERASE • 3-19, 3-22
    IO$M_INHRETRY • 3-10, 3-21, 3-22
  HSC50 controller • 3-3
  I/O functions • 3-15
    See also ACP-QIO interface
    arguments • 3-18 to 3-20
    IO$_ACPCONTROL • 1-36
    IO$_AVAILABLE • 3-24
    IO$_FORMAT • 3-22
    IO$_PACKACK • 3-23
    IO$_READLBLK • 3-20
    IO$_READPBLK • 3-20
    IO$_READVBLK • 3-20
    IO$_SEARCH • 3-23
    IO$_SEEK • 3-24
    IO$_SENSECHAR • 3-22
    IO$_SENSEMODE • 3-22
    IO$_UNLOAD • 3-24
    IO$_WRITECHECK • 3-24
    IO$_WRITELBLK • 3-21
    IO$_WRITEPBLK • 3-21
    IO$_WRITEVBLK • 3-21
  I/O status block • 3-25
  offset recovery • 3-9
  pack acknowledge function • 3-23
  port access mode • 3-7
  port selection • 3-7
  programming example • 3-25
  quotas • 1-36 to 1-38
    applicable • 3-15
  RCT (replacememt and caching table) • 3-13
  read function • 3-20
  search function • 3-23
  sector translation • 3-11
  seek operations • 3-9, 3-24
  sense mode function • 3-22
  set density function • 3-22
  skip sectoring • 3-10
  status returns • A-3
  supported devices • 3-1 to 3-6
  SYS$GETDVI • 3-14

Disk (cont'd.)
  TU58 magnetic tape • 3-6, 3-9, 3-21, 3-22, 3-23, 3-24
  UDA50 disk adapter • 3-2
  unload function • 3-24
  Verify Utility • 3-12, 3-14
  write check function • 3-24
  write function • 3-21
DISMOUNT command • 1-36
DMB32 device • 8-1
DMF32 device • 8-1
DMZ32 device • 8-1
Driver
  card reader • 2-1
  disk • 3-1
  line printer • 5-1
  LPA11-K device • 4-1
  magnetic tape • 6-1
  mailbox • 7-1
  terminal • 8-1
DSA (DIGITAL Storage Architecture)
  See DSA disk
DSA disk • 3-1, 3-8, 3-12 to 3-14
  See also Disk
  bad block replacement • 3-13
  bad blocks • 3-12
  forced error flag • 3-13
  forced errors • 3-13
  Verify Utility • 3-12, 3-14
Dual-ported disk • 3-7
  DSA disk • 3-8
  restrictions • 3-8
Duplex mode
  See also Half-duplex mode
  terminal • 8-10
DZ11 device • 8-1
DZ32 device • 8-1
DZV11 device • 8-1

E

End-of-volume
  detection on magnetic tape • 6-16
EOF (end-of-file)
  status
    card reader • 2-2
    magnetic tape • 6-13
  write mailbox message • 7-8

EOT (end-of-tape)
  status
    magnetic tape • 6-13, 6-14, 6-16
Error recovery
  disk • 3-10
  line printer • 5-3
  magnetic tape • 6-4
Escape sequence
  ANSI • B-9
  DIGITAL-private • B-9
  terminal • 8-7, 8-19
Extend subfunction • 1-12

F

FIB (file information block) • 1-3
  See also ACP function
  access control • 1-11
  contents • 1-5 to 1-8
  directory lookup • 1-9
  disk quota • 1-37 to 1-38
  extend control • 1-13
  format • 1-4
  IO$_ACCESS • 1-29
  IO$_ACPCONTROL • 1-34 to 1-38
  IO$_CREATE • 1-25
  IO$_DEACCESS • 1-31
  IO$_DELETE • 1-33
  IO$_MODIFY • 1-32
  truncate control • 1-15
File characteristics
  ACP-QIO attributes • 1-21
Floppy disk • 3-6
Form feed
  mechanical • 5-4, 8-20
Full-duplex mode • 8-10
Function code • A-1 to A-8
  See also I/O function
  IO$_ACCESS • 1-28
  IO$_ACPCONTROL • 1-34, 6-11
  IO$_AVAILABLE • 3-24, 6-21
  IO$_CREATE • 1-25
  IO$_DEACCESS • 1-31
  IO$_DELETE • 1-33
  IO$_DSE • 6-21
  IO$_FORMAT • 3-22
  IO$_INITIALIZE • 4-8
  IO$_LOADMCODE • 4-7
  IO$_MODIFY • 1-31
  IO$_PACKACK • 3-23
Function code (cont'd.)
  IO$_READLBLK • 2-4, 3-20, 6-12, 7-6, 8-25
  IO$_READPBLK • 2-4, 3-20, 6-12, 7-6
  IO$_READPROMPT • 8-25
  IO$_READVBLK • 2-4, 3-20, 6-12, 7-6, 8-25
  IO$_REWIND • 6-14
  IO$_REWINDOFF • 6-16
  IO$_SEARCH • 3-23
  IO$_SEEK • 3-24
  IO$_SENSECHAR • 3-22, 8-45
  IO$_SENSEMODE • 2-6, 3-22, 5-8, 6-17, 8-45
  IO$_SETCHAR • 2-9, 5-9, 6-18, 8-37
  IO$_SETCLOCK • 4-8
  IO$_SETMODE • 2-6, 5-9, 6-18, 8-37
  IO$_SKIPFILE • 6-14
  IO$_SKIPRECORD • 6-15
  IO$_STARTDATA • 4-9
  IO$_UNLOAD • 3-24, 6-17
  IO$_WRITECHECK • 3-24
  IO$_WRITELBLK • 3-21, 5-5, 6-13, 7-7, 8-34
  IO$_WRITEOF • 6-16
  IO$_WRITEPBLK • 3-21, 5-5, 6-13, 7-7, 8-34
  IO$_WRITEVBLK • 3-21, 5-5, 6-13, 7-7, 8-34
Function modifier • A-1 to A-8
  IO$M_ACCESS • 1-25, 1-28, 6-8
  IO$M_BINARY • 2-5
  IO$M_BRDCST • 8-45, 8-48
  IO$M_BREAKTHRU • 8-10, 8-35
  IO$M_CANCTRLO • 8-5, 8-35
  IO$M_CREATE • 1-25, 1-28, 6-8
  IO$M_CTRLCAST • 8-41
  IO$M_CTRLYAST • 8-5, 8-41
  IO$M_CVTLOW • 8-27
  IO$M_DATACHECK • 3-9, 3-21, 6-4, 6-12, 6-14
  IO$M_DELDATA • 3-22
  IO$M_DELETE • 1-25, 1-33
  IO$M_DMOUNT • 1-34
  IO$M_DSABLMBX • 8-27
  IO$M_ENABLMBX • 8-35
  IO$M_ERASE • 3-19, 3-22, 6-14
  IO$M_ESCAPE • 8-7, 8-27
  IO$M_EXTEND • 8-27, 8-28
  IO$M_HANGUP • 8-40
  IO$M_INCLUDE • 8-41, 8-43
  IO$M_INHEXTGAP • 6-5
  IO$M_INHRETRY • 3-21, 6-5
  IO$M_MAINT • 8-42
  IO$M_NOECHO • 8-10, 8-23, 8-27
  IO$M_NOFILTR • 8-27
  IO$M_NOFORMAT • 8-11, 8-35
  IO$M_NOW • 7-6, 7-7

Function modifier (cont'd.)
  IO$M_NOWAIT • 6-14, 6-16, 6-17
  IO$M_OUTBAND • 8-43
  IO$M_PACKED • 2-5
  IO$M_PURGE • 8-27
  IO$M_RD_MODEM • 8-47
  IO$M_READATTN • 7-8
  IO$M_REFRESH • 8-35
  IO$M_REVERSE • 6-12
  IO$M_SET_MODEM • 8-42
  IO$M_SETEVF • 4-9
  IO$M_SETPROT • 7-10
  IO$M_TIMED • 8-27
  IO$M_TRMNOECHO • 8-28
  IO$M_TT_ABORT • 8-43
  IO$M_TYPEAHDCNT • 8-46
  IO$M_UNLOOP • 8-43

H

Half-duplex mode • 8-10, 8-19
  See also Duplex mode
Hang up
  function modifier • 8-40
  terminal • 8-16, 8-23
HSC50 disk controller • 3-3

I

I/O function
  See also Function code
  ACP-QIO interface • 1-2
  arguments • A-1 to A-8
  card reader • 2-4
  codes • A-1
  disk • 1-2, 3-15
  line printer • 5-5
  LPA11-K device • 4-6
  magnetic tape • 1-2, 6-8
  modifiers • A-1 to A-8
  terminal • 8-25
I/O status block
  ACP-QIO interface • 1-39
  card reader • 2-9
  disk • 3-25
  line printer • 5-10
  LPA11-K device • 4-31
  magnetic tape • 6-22
I/O status block (cont'd.)
  mailbox • 7-11
  terminal • 8-48
INITIALIZE command • 6-21
Initialize command table
  LPA11-K device • 4-8

K

Keyboard control character • 8-3 to 8-6, 8-9

L

Laboratory Peripheral Accelerator
  See LPA11-K device
LAT line • 8-1
Line printer
  carriage control • 5-6, 5-7
  character case • 5-4
  character formatting • 5-2
  device characteristics • 5-3
  driver • 5-1
  error recovery • 5-3
  function codes • 5-5, A-5
  I/O functions
    IO$_SENSEMODE • 5-8
    IO$_SETCHAR • 5-9
    IO$_SETMODE • 5-9
    IO$_WRITELBLK • 5-5
    IO$_WRITEPBLK • 5-5
    IO$_WRITEVBLK • 5-5
  I/O status block • 5-10
  mechanical form feed • 5-4
  printall mode • 5-4
  programming example • 5-10
  sense mode function • 5-8
  set characteristics • 5-9
  set mode function • 5-9
  status returns • A-5
  supported devices • 5-1
  SYS$GETDVI • 5-3
  write function • 5-5
    carriage control • 5-6
Line terminator, terminal • 8-9
LPA11-K device
  AST
    address • 4-10, 4-12
    quota • 4-12
    synchronization • 4-12

LPA11-K device (cont'd.)
  buffer management • 4-14
  buffer overrun • 4-10, 4-12, 4-29
  buffer queue control • 4-13
  clock rate • 4-8
  data buffer • 4-11
  data sampling • 4-1
  data transfer command table • 4-10
  data transfer start command • 4-10
  data transfer stop command • 4-12
  data underrun/overrun • 4-10
  device characteristics • 4-3 to 4-6
  device configuration • 4-1, 4-8, 4-32
  device initialization • 4-3, 4-7 to 4-8, 4-31, 4-32
  driver • 4-1
  errors • 4-2
  features • 4-2
  function codes • 4-6, A-3
  function modifier
    IO$M_SETEVF • 4-9, 4-12
  high-level language support routines • 4-13
  I/O functions
    IO$_INITIALIZE • 4-8
    IO$_LOADMCODE • 4-7
    IO$_SETCLOCK • 4-8
    IO$_STARTDATA • 4-9
    IO$_STARTMPROC • 4-7
  I/O status block • 4-31
  initialize command table • 4-8
  initialize function • 4-8
  load microcode function • 4-7
  maintenance status register • 4-8, 4-31
  microcode loading • 4-3, 4-7, 4-31, 4-32
  modes of operation • 4-1
  operator process • 4-33
  programming examples • 4-35, 4-37, 4-41
  RSX–11M/M–PLUS and VAX/VMS differences • 4-33
  set clock function • 4-8
  start data transfer request function • 4-9
  start microprocessor function • 4-7
  status returns • 4-7, 4-8, 4-9, 4-11, 4-31, A-5
  stop command • 4-12
  subroutines
    argument usage • 4-14 to 4-18
    list • 4-13
    LPA$ADSWP • 4-18
    LPA$CLOCKA • 4-24
    LPA$CLOCKB • 4-25
    LPA$CVADF • 4-30
    LPA$DASWP • 4-19
    LPA$DISWP • 4-19

LPA11-K device
  subroutines (cont'd.)
    LPA$DOSWP • 4-20
    LPA$FLT16 • 4-30
    LPA$IBFSTS • 4-26
    LPA$IGTBUF • 4-27
    LPA$INXTBUF • 4-28
    LPA$IWTBUF • 4-28
    LPA$LAMSKS • 4-21
    LPA$LOADMC • 4-31
    LPA$RLSBUF • 4-29
    LPA$RMVBUF • 4-30
    LPA$SETADC • 4-22
    LPA$SETIBF • 4-22
    LPA$STPSWP • 4-23
    LPA$XRATE • 4-25
  supported device • 4-1
  supporting software • 4-2
  SYS$CANCEL • 4-12
  SYS$GETDVI • 4-3

M

Magnetic tape
  ACP control function • 1-34, 6-11
  ACP create file operation • 1-28
  available function • 6-21
  BOT marker • 6-14, 6-15
  byte count
    read • 6-13
    write • 6-14
  capabilities • 6-3
  data check • 6-4, 6-12, 6-14
  data security erase function • 6-21
  density • 6-20
  device characteristics • 6-6 to 6-7
  driver • 6-1
  end-of-file status • 6-13
  end-of-volume detection • 6-16
  EOF status • 6-13
  EOT
    marker • 6-15 to 6-16
    status • 6-13, 6-14, 6-16
  error recovery • 6-4
  extended characteristics • 6-7
  file attributes • 6-4
  function codes • 6-8, A-5
  function modifiers
    IO$M_DATACHECK • 6-4, 6-12, 6-14
    IO$M_ERASE • 6-14
    IO$M_INHEXTGAP • 6-5

Magnetic tape
  function modifiers (cont'd.)
    IO$M_INHRETRY • 6-5
    IO$M_NOWAIT • 6-14, 6-16, 6-17
    IO$M_REVERSE • 6-12
  I/O functions • 6-8
    See also ACP-QIO interface
    arguments • 6-11
    IO$_ACCESS • 6-8
    IO$_ACPCONTROL • 1-35, 6-11
    IO$_AVAILABLE • 6-21
    IO$_CREATE • 6-8
    IO$_DEACCESS • 6-8
    IO$_DSE • 6-8, 6-21
    IO$_MODIFY • 6-8
    IO$_PACKACK • 6-21
    IO$_READLBLK • 6-12
    IO$_READPBLK • 6-12
    IO$_READVBLK • 6-12
    IO$_REWIND • 6-14
    IO$_REWINDOFF • 6-16
    IO$_SENSEMODE • 6-17
    IO$_SETCHAR • 6-18
    IO$_SETMODE • 6-18
    IO$_SKIPFILE • 6-14
    IO$_SKIPRECORD • 6-15
    IO$_UNLOAD • 6-17
    IO$_WRITELBLK • 6-13
    IO$_WRITEOF • 6-16
    IO$_WRITEPBLK • 6-13
    IO$_WRITEVBLK • 6-13
  I/O status block • 6-22
  master adapters • 6-3
  pack acknowledge function • 6-21
  parity • 6-20
  positioning • 1-35
  programming example • 6-22
  quotas • 6-8
  read function • 6-12
  read reverse function • 6-12, 6-13
  rewind function • 6-14
  rewind offline function • 6-16
  sense mode function • 6-17
  set characteristics function • 6-18
  set mode function • 6-18
    characteristics • 6-20
  skip file function • 6-14
  skip record function • 6-15
  slave formatter • 6-3
  status returns • A-6
  streaming tape systems • 6-5
  supported devices • 6-1
Magnetic tape (cont'd.)
  SYS$GETDVI • 6-6
  tape controllers • 6-2
  tape mark • 6-13, 6-15, 6-16
  thrashing • 6-5
  TMSCP mmagnetic tapes • 6-1
  TU58 magnetic tape
    See Disk, TU58
  unload function • 6-17
  write end-of-file function • 6-16
  write function • 6-13
Mailbox
  See also Terminal
  creating • 7-2
  deleting • 7-3
  device characteristics • 7-5
  disable terminal • 8-20
  driver • 7-1
  explanation • 7-1
  function codes • 7-6, A-7
  function modifiers
    IO$M_NOW • 7-4, 7-6, 7-7, 7-8, 7-9
    IO$M_READATTN • 7-8
    IO$M_SETPROT • 7-10
  I/O functions
    IO$_READLBLK • 7-6
    IO$_READPBLK • 7-6
    IO$_READVBLK • 7-6
    IO$_WRITELBLK • 7-7
    IO$_WRITEOF • 7-8
    IO$_WRITEPBLK • 7-7
    IO$_WRITEVBLK • 7-7
  I/O status block • 7-11
  message format • 7-4
    terminal • 8-17
  message size • 7-2
  multiport memory • 7-1
  permanent • 7-2, 7-4
  programming example • 7-12
  protection • 7-2, 7-4, 7-10
  read attention AST function • 7-8
  read function • 7-6
  set attention AST function • 7-8
  set protection function • 7-10
  status returns • A-7
  SYS$GETDVI • 7-5
  temporary • 7-2, 7-4
  terminal/mailbox interaction • 8-16
  volume protection • 7-10
  write attention AST function • 7-8
  write end-of-file message function • 7-8
  write function • 7-7

Master adapter • 6-3
Mechanical form feed
  line printer • 5-4
  terminal • 8-20
Message format
  See Mailbox
Modify file function • 1-31
MOUNT command • 6-21
Mount function • 1-33
Multinational character set
  See DEC Multinational Character Set
Multiplexer
  DMB32 device • 8-1
  DMF32 device • 8-1
  DZ11 device • 8-1
  DZ32 device • 8-1

O

Out-of-Band AST • 8-43
Out-of-band AST • 8-11

P

Parity flag • 8-39
Passall mode • 5-4
Pasthru mode • 8-9, 8-11, 8-24, 8-26
Permanent mailbox • 7-4
Port access mode • 3-7
Port selection • 3-7
Printer
  See Line printer
Protection
  mailbox • 7-2, 7-4

Q

Quota
  AST • 3-15, 4-12, 6-8, 7-6, 7-9, 8-41
  buffered I/O • 3-15, 6-8, 7-6
  BYTELIM • 1-12
  direct I/O • 3-15, 6-8
  disk • 1-36 to 1-38
  mailbox buffer • 7-2, 7-4, 7-6
Quota file transfer block • 1-37

R

Read/write attributes subfunction • 1-16
Read attention AST function • 7-8
Record attributes value • 1-21
RETURN key • 8-6
Rewind offline function • 6-16
RSX-11M/M-PLUS
  VAX/VMS LPA11-K routine differences • 4-33
RX01 console disk • 3-5

S

Sector translation • 3-11
Seek operation • 3-9
Sense tape mode function • 6-17
Serial line multiplexer • 8-1
Set attention AST
  See Attention AST
Set characteristic
  card reader • 2-6
  line printer • 5-9
  magnetic tape • 6-18
  terminal • 8-37
Set mode
  card reader • 2-6
  line printer • 5-9
  magnetic tape • 6-18
  mailbox • 7-8
  terminal • 8-37
SET TERMINAL DCL command • 8-3, 8-17, 8-24
Set translation mode • 2-2
Skip file function • 6-15
Skip sectoring • 3-10
Slave formatter • 6-3
SS$_ABORT • 8-42, A-2, A-3, A-5, A-6, A-7, A-9
SS$_ACCONFLICT • A-1
SS$_ACCVIO • 7-11
SS$_ACPVAFUL • A-1
SS$_BADATTRIB • A-1
SS$_BADCHKSUM • A-1
SS$_BADESCAPE • 8-7, A-9
SS$_BADFILEHDR • A-1
SS$_BADFILENAME • A-1
SS$_BADFILEVER • A-1
SS$_BADIRECTORY • A-1
SS$_BADPARAM • A-1, A-5, A-9
SS$_BADQFILE • A-1
SS$_BLOCKCNTERR • A-1

SS$_BUFFEROVF • 7-6, A-7
SS$_BUFNOTALIGN • A-5
SS$_CANCEL • A-3, A-5, A-6, A-9
SS$_CONTROLC • 8-43, A-9
SS$_CONTROLO • A-9
SS$_CONTROLY • A-9
SS$_CREATED • A-1
SS$_CTRLERR • A-3, A-5, A-6
SS$_DATACHECK • A-3, A-5, A-6
SS$_DATAOVERUN • 8-9, A-2, A-3, A-6, A-9
SS$_DEVACTIVE • A-5
SS$_DEVCMDERR • A-5
SS$_DEVICEFULL • A-1
SS$_DEVOFFLINE • A-6
SS$_DEVREQERR • A-5
SS$_DIRFULL • A-1
SS$_DIRNOTEMPTY • A-1
SS$_DRVERR • A-3, A-6
SS$_DUPDSKQUOTA • A-1
SS$_DUPFILENAME • A-1
SS$_ENDOFFILE • 6-16, 7-6, 7-8, A-1, A-2, A-6, A-7
SS$_ENDOFTAPE • A-6
SS$_ENDOFVOLUME • 6-16, A-6
SS$_EXBYTLM • A-1
SS$_EXDISKQUOTA • A-1
SS$_EXQUOTA • A-5
SS$_FCPREADERR • A-1
SS$_FCPREWNDERR • A-1
SS$_FCPSPACERR • A-1
SS$_FCPWRITERR • A-1
SS$_FILELOCKED • A-1
SS$_FILENUMCHK • A-1
SS$_FILEPURGED • A-1
SS$_FILESEQCHK • A-1
SS$_FILESTRUCT • A-1
SS$_FILNOTEXP • A-1
SS$_FORCEDERR • A-3
SS$_FORMAT • A-3, A-6
SS$_HANGUP • 8-11
SS$_HEADERFULL • A-1
SS$_IBCERROR • A-1
SS$_IDXFILEFULL • A-1
SS$_ILLCNTRFUNC • A-1
SS$_ILLIOFUNC • A-3, A-6
SS$_INCOMPAT • A-9
SS$_INSFBUFDP • A-5
SS$_INSFMAPREQ • A-5
SS$_INSFMEM • 7-11, A-5
SS$_IVADDR • A-3
SS$_IVBUFLEN • A-3, A-5
SS$_IVMODE • A-5
SS$_MBFULL • 7-3, 7-11
SS$_MBTOOSML • 7-11
SS$_MCNOTVALID • A-5
SS$_MEDOFL • A-3, A-6
SS$_NODISKQUOTA • A-1
SS$_NOMOREFILES • A-1
SS$_NONEXDRV • A-3, A-6
SS$_NOPRIV • 7-11, A-1
SS$_NOQFILE • A-1
SS$_NORMAL • A-2, A-3, A-6, A-7, A-9
SS$_NOSUCHFILE • A-1
SS$_NOTAPEOP • A-2
SS$_NOTLABELMT • A-2
SS$_NOTPRINTED • A-2
SS$_NOTVOLSET • A-2
SS$_OPINCOMPL • A-3, A-6
SS$_OVRDSKQUOTA • A-2
SS$_PARITY • A-3, A-5, A-6, A-9
SS$_PARTESCAPE • 8-7, 8-30, A-9
SS$_POWERFAIL • A-5
SS$_QFACTIVE • A-2
SS$_QFNOTACT • A-2
SS$_RCT • A-3
SS$_RDDELDATA • A-3
SS$_SUPERSEDE • A-2
SS$_TAPEPOSLOST • A-2
SS$_TIMEOUT • 8-27, A-3, A-5, A-6, A-9
SS$_TOOMANYVER • A-2
SS$_UNSAFE • A-3, A-6
SS$_VOLINV • A-3, A-6
SS$_WASECC • A-3
SS$_WRITLCK • A-2, A-3, A-6
SS$_WRONGACP • A-2
SYS$ASSIGN • 7-2, 8-16
SYS$CANCEL • 4-12
SYS$CREMBX • 7-2
SYS$DASSGN • 7-3
SYS$DELMBX • 7-4
SYS$DISMOUNT • 1-36
SYS$GETDVI • 6-6
  card reader • 2-3
  disk • 3-14
  line printer • 5-3
  LPA11-K device • 4-3
  mailbox • 7-5
  terminal • 8-18
System console terminal • 8-1

T

- Tab
  - CTRL/I • 8-6
  - terminal mechanical • 8-20
  - terminal tab stops • 8-34
- Tape
  - See Magnetic tape
- Tape mark • 6-13, 6-15, 6-16
- Temporary mailbox • 7-4
- Terminal
  - ANSI CRT terminal • 8-21
  - autobaud detection • 8-17, 8-21
  - baud rate • 8-17, 8-21, 8-39
  - bell (CTRL/G) • 8-9
  - broadcast message • 8-16, 8-20, 8-22, 8-45
  - carriage control • 8-35
  - characteristic
    - See Terminal characteristic
  - command line editing • 8-3, 8-34
  - command recall (CTRL/B) • 8-3, 8-6
  - control and data signals • 8-12
  - control characters • 8-3 to 8-6, 8-9, 8-26
    - numeric values • B-1
  - control sequences • 8-8
  - cursor movement • 8-3, 8-5, 8-21
  - delete character • 8-3
  - delete line (CTRL/U) • 8-4, 8-26
  - device characteristics • 8-18, 8-19
    - categories • 8-24
    - changing • 8-40
    - extended • 8-21
  - dial-up
    - characteristic • 8-20
    - lines • 8-11, 8-22, 8-40
    - support • 8-11
  - DIGITAL CRT terminal • 8-22
  - discard output (CTRL/O) • 8-5, 8-26, 8-35
  - driver • 8-1
  - duplex modes • 8-10, 8-11
  - enable CTRL/C AST • 8-41
  - enable CTRL/Y AST • 8-41
  - escape sequences • 8-7, 8-48
    - ANSI • B-9
    - DIGITAL-private • B-9
    - overflow size (item code) • 8-30
  - extended characteristics • 8-21
  - features and capabilities • 8-2
  - form feed • 8-20, 8-34
  - frame size • 8-40
  - function codes • 8-25, A-7

- Terminal (cont'd.)
  - function modifiers
    - See also Terminal, item codes
    - IO$M_BRDCST • 8-45, 8-48
    - IO$M_BREAKTHRU • 8-10, 8-35
    - IO$M_CANCTRLO • 8-5, 8-35
    - IO$M_CTRLCAST • 8-41
    - IO$M_CTRLYAST • 8-5, 8-11, 8-41
    - IO$M_CVTLOW • 8-27
    - IO$M_DSABLMBX • 8-27
    - IO$M_ENABLMBX • 8-35
    - IO$M_ESCAPE • 8-7, 8-27
    - IO$M_EXTEND • 8-27, 8-28
    - IO$M_HANGUP • 8-40
    - IO$M_INCLUDE • 8-18, 8-41, 8-43
    - IO$M_LOOP • 8-42
    - IO$M_MAINT • 8-42
    - IO$M_NOECHO • 8-9, 8-10, 8-23, 8-27
    - IO$M_NOFILTR • 8-27
    - IO$M_NOFORMAT • 8-11, 8-35, 8-43
    - IO$M_OUTBAND • 8-43
    - IO$M_PURGE • 8-27
    - IO$M_RD_MODEM • 8-47
    - IO$M_REFRESH • 8-35
    - IO$M_SET_MODEM • 8-42
    - IO$M_TIMED • 8-27
    - IO$M_TRMNOECHO • 8-28
    - IO$M_TT_ABORT • 8-18, 8-43
    - IO$M_TYPEAHDCNT • 8-46
    - IO$M_UNLOOP • 8-43
  - hang up • 8-11, 8-14, 8-16, 8-23, 8-40
  - I/O functions
    - IO$_READLBLK • 8-25
    - IO$_READPROMPT • 8-25, 8-26
    - IO$_READVBLK • 8-25
    - IO$_SENSECHAR • 8-45
    - IO$_SENSEMODE • 8-45
    - IO$_SETCHAR • 8-37
    - IO$_SETMODE • 8-37
    - IO$_WRITELBLK • 8-34
    - IO$_WRITEPBLK • 8-34
    - IO$_WRITEVBLK • 8-34
  - I/O status block • 8-48
  - initiating login • 8-9
  - input processing • 8-2
  - insert/overstrike (CTRL/A) • 8-3, 8-6
  - interrupt (CTRL/Y) • 8-5
  - item codes • 8-30 to 8-33
  - itemlist read • 8-28
    - example • 8-50
    - item codes • 8-30 to 8-33
    - item descriptor • 8-29

Terminal (cont'd.)
  - line editing • 8-3, 8-23
    - See also Terminal, item codes
  - line feed • 8-34
  - line terminators • 8-9
  - mailbox • 8-16, 8-35
    - message format • 8-17
    - message types • 8-16
  - modem
    - characteristic • 8-20
    - control signals • 8-12
    - data signals • 8-12
    - protocol • 8-12
    - sense signals • 8-47
    - signal control • 8-11
  - no type-ahead • 8-20
  - out-of-band
    - AST • 8-11, 8-41, 8-43
    - characters • 8-18
  - output formatting • 8-11, 8-24
  - output processing • 8-10
  - page length and width • 8-38, 8-46
  - parity flag • 8-39
  - pasthru mode • 8-9, 8-11, 8-24, 8-26
  - process preservation • 8-14
  - programming examples • 8-50
  - protocol • 8-12
  - read function • 8-25
    - arguments • 8-26
    - function modifiers • 8-27
    - itemlist read • 8-28
    - terminating • 8-26, 8-27
    - terminators • 8-28
    - with timeout • 8-26, 8-27
  - read verify • 8-7, 8-33
    - example • 8-50
  - receive speed • 8-39
  - redisplay data (CTRL/R) • 8-6, 8-26
  - ReGIS graphics • 8-24
  - restart data (CTRL/Q) • 8-6
  - sense characteristics function • 8-45
  - sense mode function • 8-45
  - serial line multiplexer • 8-1
  - set characteristics function • 8-37
    - arguments • 8-37
  - set mode function • 8-37
    - arguments • 8-37
  - SET TERMINAL DCL command • 8-3, 8-17, 8-24
  - SIXEL graphics • 8-24
  - special operating modes • 8-10
  - status (CTRL/T) • 8-6

Terminal (cont'd.)
  - status returns • A-9
  - stop data (CTRL/S) • 8-6
  - supported devices • 8-1
  - SYS$GETDVI • 8-18
  - system password • 8-24
  - tab
    - CTRL/I • 8-6
    - mechanical • 8-20
    - stops • 8-34
  - terminator mask • 8-28
  - time (CTRL/T) • 8-6
  - transmit speed • 8-39
  - TTY_DIALTYPE SYSGEN parameter • 8-11, 8-12, 8-14
  - type-ahead • 8-8, 8-16, 8-19, 8-46
    - alternate buffer • 8-21
  - unsolicited data • 8-16
  - write breakthrough function • 8-35
  - write function • 8-34
    - carriage control • 8-35
    - function modifiers • 8-35
  - XON/XOFF control • 8-24

Terminal characteristic
  - ANSI CRT • 8-21
  - ASCII (8-bit) code • 8-19
  - baud rate • 8-21
  - block mode • 8-22
  - dial-up line • 8-23
  - dial-up terminal • 8-20
  - DIGITAL CRT • 8-22
  - DMA mode • 8-23
  - edit • 8-23
  - extended characteristics • 8-21
  - local echo • 8-23
  - modem • 8-20
  - modify hang up • 8-24
  - no echo • 8-20
  - no type ahead • 8-20
  - pasthru mode • 8-24
  - ReGIS graphics • 8-24
  - remote terminal • 8-20
  - secure • 8-24
  - set speed • 8-24
  - SIXEL graphics • 8-24
  - system password • 8-24
  - XON/XOFF • 8-24

Terminator character bit mask • 8-28

Thrashing
  - magnetic tape • 6-5

Translation
  - logical to physical • 3-11

Truncate subfunction • 1-15

TU58 magnetic tape
  See Disk
Type-ahead
  terminal • 8-8, 8-21, 8-46

U

UDA50 disk adapter • 3-2
Unload function
  disk • 3-24
  magnetic tape • 6-17

W

Write attention AST function • 7-9
Write breakthrough function • 8-35
Write end-of-file function
  magnetic tape • 6-16
  message • 7-8

X

XQP (extended QIO processor) • 1-1

VAX/VMS I/O User's
Reference Manual: Part I
AA–Z600C–TE

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code or Country ______

— — Do Not Tear - Fold Here and Tape — — — — — — — — — — — — — — — — — — —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

— — Do Not Tear - Fold Here —

Cut Along Dotted Line

VAX/VMS I/O User's
Reference Manual: Part I
AA–Z600C–TE

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code ______
or Country

- — Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

- — Do Not Tear - Fold Here

Cut Along Dotted Line