DISK OPERATING SYSTEM DOS.

User's Guide

Version 2 (Upgraded to 2.6)

May, 1980

Document No. 50432

DATAPOINT

DISK OPERATING SYSTEM
DOS.

User's Guide

Version 2 (Upgraded to 2.6)

May, 1980

Document No. 50432

PREFACE

The purpose of this User's Guide is to provide the user of a Datapoint DOS that information required to generate a system, make effective use of the available commands, and to make user-written programs compatible with the DOS.

This manual applies to all Version 2.6 and above "dot-series" Disk Operating Systems, such as DOS.A, DOS.B, and so on. This manual replaces the previous Version 2.5 DOS User's Guide.

TABLE OF CONTENTS

	page
1. CHANGES FROM PRIOR VERSIONS	1-1
1.1 Utility Programs	1-1
1.1.1 BACKUP	1-1
1.1.2 CAT	1-1
1.1.3 COPY	1-1
1.1.4 DOSGEN	1-1
1.1.5 DUMP	1-2
1.1.6 EDIT	1-2
1.1.7 INDEX	1-2
1.1.8 MIN	1-2
1.1.9 PUTIPL	1-2
1.1.10 PUTVOLID	1-2
1.1.11 REFORMAT	1-2
1.1.12 SORT	1-3
1.2 System Routines	1-3
2. INTRODUCTION	2-1
2.1 Hardware Support Required	2-1
2.2 Software Configurations Available	2-2
2.3 Program Compatibility	2-2
3. OPERATOR COMMANDS	3-1
3.1 General Information	3-1
3.2 Command Line Syntax	3-1
3.3 Command Interpretation	3-2
3.4 Documentation Conventions	3-3
3.5 Program Signon Messages	3-3
4. EQUIPMENT CARE	4-1
4.1 Environment	4-1
4.2 Processor	4-2
4.3 Disks and Disk Drives	4-2
4.4 Other Peripherals	4-2
5. DISK FILES	5-1
5.1 File Names	5-1
5.2 File Creation	5-2
5.3 File Deletion	5-3
5.4 File Protection	5-3
6. SYSTEM GENERATION	6-1
6.1 Initial Generation	6-1
6.1.1 Formatting	6-1

6.1.2 Cassette System Generation 6-2
6.2 Partial Generation 6-4
6.3 UPGRADE/X 6-5
6.4 Scratch Disk Generation 6-6
6.5 Generation Cassettes and Emergencies 6-6

7. ABTONOFF COMMAND 7-1
7.1 Purpose 7-1
7.2 Use 7-1

8. APP COMMAND 8-1
8.1 Purpose 8-1
8.2 Use 8-1

9. AUTO COMMAND 9-1
9.1 Purpose 9-1
9.2 Use 9-1
9.3 Operation of AUTOed Program 9-2

10. AUTOKEY COMMAND 10-1
10.1 Purpose 10-1
10.2 Use 10-1
10.3 The Hardware Auto-Restart Facility 10-2
10.3.1 Processors with tape decks 10-2
10.3.2 Processors without tape decks 10-2
10.4 Automatic Program Execution Using AUTO 10-3
10.5 Auto-Restart Facilities Using AUTOKEY 10-3
10.6 A Simple Example 10-3
10.7 A More Complicated Example 10-4
10.8 Special Considerations 10-8
10.9 AUTOKEY and DATASHARE 10-8

11. BACKUP COMMAND 11-1
11.1 Purpose 11-1
11.2 Use 11-1
11.2.1 Options 11-2
11.3 Mirror Image Copy 11-3
11.4 Reorganizing Files 11-3
11.4.1 Copying DOS to Output Disk 11-4
11.4.2 Deleting Named Files 11-4
11.4.3 Copying Named Files 11-4
11.5 Use of KEYBOARD and DISPLAY Keys 11-5
11.6 Error Messages 11-5
11.7 Reorganizing Files for Faster Processing 11-6
11.8 BACKUP with CHAIN 11-7
11.9 Clicks during Copying 11-7
11.10 Special Considerations for BACKUP 11-7

12. BLOKEDIT COMMAND 12-1
12.1 Purpose 12-1
12.2 Use 12-1
12.3 File Descriptions 12-2
12.3.1 Command Statement Lines 12-2
12.3.2 Source File 12-4
12.3.3 New File 12-4
12.4 Messages 12-4
12.4.1 Informative Messages 12-6
12.4.2 Fatal Errors 12-6
12.4.3 Selectively Fatal Errors 12-7

13. BOOT 13-1
13.1 Purpose 13-1
13.2 Use 13-1
13.3 Messages 13-2

14. BUILD COMMAND 14-1
14.1 Purpose 14-1
14.2 Use 14-1
14.3 A Simple Example 14-2
14.4 KEYBOARD and DISPLAY Keys 14-2

15. CAT COMMAND 15-1
15.1 Purpose 15-1
15.2 Use 15-1

16. CHAIN COMMAND 16-1
16.1 Purpose 16-1
16.2 Use 16-1
16.2.1 CHAIN Compilation 16-2
16.2.2 CHAIN Execution 16-2
16.3 Tag Definition 16-3
16.4 CHAIN Directives 16-4
16.4.1 IF Directive 16-4
16.4.2 ELSE/XIF Directives 16-6
16.5 Tag Value Substitution 16-6
16.6 BEGIN/END Directives 16-7
16.7 ABORT Directives 16-8
16.8 Comments 16-9
16.9 Complex CHAIN Examples 16-11
16.10 Resuming An Aborted CHAIN 16-14
16.11 CHAIN Programming Considerations 16-15

17. CHANGE COMMAND 17-1
17.1 Purpose 17-1
17.2 Use 17-1

18. COPY COMMAND 18-1
18.1 Purpose 18-1
18.2 Use 18-1

19. DOSGEN COMMAND 19-1
19.1 Purpose 19-1
19.2 Use 19-1
19.3 Special Considerations 19-2

20. DSKCHECK 20-1
20.1 Purpose 20-1
20.2 Use 20-1
20.3 Options 20-1
20.4 System Tables and Data 20-2
20.5 Execution Phases 20-2
20.5.1 Initialization 20-2
20.5.2 HDI Checking 20-3
20.5.3 CAT Checking 20-3
20.5.4 Directory Checking 20-3
20.5.5 RIB Checking 20-4
20.5.6 Cluster Allocation, Phase 1 20-5
20.5.7 Cluster Allocation, Phase 2 20-5
20.5.8 Lockout CAT Checking 20-5
20.6 Operational Messages 20-5
20.7 Error Message Definitions 20-7
20.7.1 Operational Error Messages 20-7
20.7.2 Initialization Error Messages 20-8
20.7.3 HDI Errors 20-10
20.7.4 CAT Errors 20-10
20.7.5 Directory Errors 20-12
20.7.6 RIB Errors 20-14
20.7.7 Lockout CAT Errors 20-17

21. DUMP COMMAND 21-1
21.1 Purpose 21-1
21.2 Use 21-1
21.3 Informational Messages Provided 21-2
21.4 Level One Commands To DUMP 21-4
21.5 Level Two Commands To DUMP 21-4
21.6 Level Three Commands To DUMP 21-5
21.7 Level Four Commands To DUMP 21-6
21.8 Level Five Commands to DUMP 21-7
21.9 Error Messages 21-7

22. THE DUMP93X0 COMMAND 22-1
22.1 Purpose 22-1
22.2 Use 22-1
22.3 The primary command handler 22-3

22.4 Using 22-3
22.5 Screen Display format 22-5
22.6 The Screen Dump Command Handler 22-6
22.7 Cassette Operations 22-8
22.8 Drive Numbers 22-10
22.9 Error Messages 22-10

23. EDIT 23-1
23.1 Purpose 23-1
23.2 Use 23-1
23.3 Parameter List 23-2
23.3.1 Margin Bell 23-2
23.3.2 Tab Key Character 23-3
23.3.3 Mode 23-3
23.3.4 Update 23-3
23.3.5 Key-click 23-4
23.3.6 Space Compression 23-4
23.3.7 Non-verification 23-4
23.4 Examples 23-4
23.5 Data Entry and Retrieval 23-6
23.5.1 Data Entry 23-6
23.5.2 Multi-line Record Entry 23-7
23.6 Data Retrieval 23-8
23.7 EDIT Command Format 23-8
23.8 Basic EDIT Commands 23-9
23.8.1 Setting Tabs 23-9
23.8.2 Setting TEXT Mode 23-9
23.8.3 INSERTing a Line 23-10
23.8.4 DELETEing a Line 23-11
23.8.5 COPYing a Line 23-11
23.8.6 MODIFYing a Line 23-12
23.8.7 LOCATEing a Line 23-13
23.8.8 ENDing EDIT 23-13
23.9 Intermediate Commands 23-14
23.9.1 Changing Special Characters 23-14
23.9.2 Changing Tabs 23-15
23.9.3 Changing Modes and Options 23-16
23.9.4 Deleting Lines 23-17
23.9.5 MODIFY Command 23-18
23.9.5.1 Line Modification 23-18
23.9.5.2 Field Modification 23-20
23.9.6 Line Splitting 23-21
23.9.7 Line Concatenation 23-21
23.9.8 File Search Commands 23-21
23.9.9 BYPASS End of File 23-24
23.9.10 Terminating EDIT 23-24
23.10 Advanced Commands 23-26
23.11 Recovery Procedures 23-30

23.12 Glossary 23-30
23.13 Command List 23-34
23.14 EDIT ERROR MESSAGES 23-41
23.15 Configuration Sector 23-42
23.16 Example of a Definition File 23-44

24. ENCODE/DECODE COMMANDS 24-1
24.1 Purpose 24-1
24.2 Use 24-1

25. FILES COMMAND 25-1
25.1 Purpose 25-1
25.2 Use 25-1
25.3 Default Messages 25-2
25.4 File Descriptions 25-3
25.5 Error Messages 25-4

26. FIX COMMAND 26-1
26.1 Purpose 26-1
26.2 Use 26-1
26.3 Commands 26-1
26.4 Error Messages 26-3

27. FIXAPPLY 27-1
27.1 Purpose 27-1
27.2 Use 27-1
27.2.1 FIXAPPLY Phase One 27-1
27.2.2 FIXAPPLY Phase Two 27-2
27.2.3 Fatal Phase One Error Messages 27-3

28. FREE COMMAND 28-1
28.1 Purpose 28-1
28.2 Use 28-1

29. INDEX COMMAND 29-1
29.1 Purpose 29-1
29.2 Use 29-1
29.2.1 Parameters 29-2
29.2.2 System Requirements 29-3
29.3 Choosing A Record Key 29-3
29.4 Preprocessing the File 29-4
29.4.1 Invoking Reformat 29-4
29.4.2 Considerations for Unattended Indexing 29-5
29.5 INDEX Messages 29-5
29.6 ISI File Formats 29-7
29.7 Index File Size 29-10
29.8 Examples of the Use of INDEX 29-11

30. THE INITDISK COMMAND 30-1
30.1 Purpose 30-1
30.2 Use 30-1
30.3 Error messages 30-1

31. KILL COMMAND 31-1
31.1 Purpose 31-1
31.2 Use 31-1

32. LIST COMMAND 32-1
32.1 Purpose 32-1
32.2 Use 32-1
32.3 Input File Specification 32-2
32.4 Starting Point 32-2
32.5 Output File Specification 32-3
32.6 Output Device 32-3
32.7 Output Format 32-4
32.8 Format Control 32-4
32.9 Operator Controls 32-5
32.10 Error Conditions 32-5

33. MANUAL COMMAND 33-1
33.1 Purpose 33-1
33.2 Use 33-1

34. MIN COMMAND 34-1
34.1 Purpose 34-1
34.2 Use 34-1
34.2.1 Command Line 34-1
34.2.2 Options 34-1
34.2.3 Multi-File Named Tapes 34-3
34.2.3.1 MOUT With Directory Tapes 34-3
34.2.3.2 CTOS Tapes 34-5
34.2.4 Multiple Numbered-File Tapes 34-6
34.2.5 Double File Tapes 34-6
34.2.6 Single File Tapes 34-7
34.3 Tape Formats 34-7
34.3.1 Single File Tapes 34-7
34.3.2 Double File Tapes 34-8
34.3.3 Multiple Numbered-File Tapes 34-8
34.3.4 Multiple Named-File Tapes 34-8
34.4 Errors 34-9

35. MOUT COMMAND 35-1
35.1 Purpose 35-1
35.2 Use 35-1
35.3 File Names 35-5
35.4 Writing 35-7

35.5 Verifying 35-8

36. NAME COMMAND 36-1
36.1 Purpose 36-1
36.2 Use 36-1

37. PUTIPL COMMAND 37-1
37.1 Purpose 37-1
37.2 Use 37-1

38. PUTVOLID COMMAND 38-1
38.1 Purpose 38-1
38.2 Use 38-1

39. REFORMAT COMMAND 39-1
39.1 Purpose 39-1
39.2 Use 39-1
39.3 Output File Formats 39-3
39.4 Reasons for Reformatting 39-3
39.5 Reformat Messages 39-4
39.6 Text File Formats 39-7

40. REWIND COMMAND 40-1
40.1 Purpose 40-1
40.2 Use 40-1

41. SAPP COMMAND 41-1
41.1 Purpose 41-1
41.2 Use 41-1

42. SORT COMMAND 42-1
42.1 Purpose 42-1
42.2 Use 42-1
42.3 Fundamental SORT Concepts 42-2
42.3.1 File Formats 42-2
42.3.2 The Key Options 42-3
42.3.3 How to Sort a File 42-4
42.4 SORT Command Line and Options 42-4
42.4.1 Generalized Command Statement Format 42-4
42.4.2 Keys: Overlapping and in Backwards Order 42-9
42.4.3 Collating Sequence File 42-10
42.4.4 Ascending and Descending sequences 42-11
42.4.5 Input/output File Format Options 42-11
42.4.6 Limited Output Format Option 42-12
42.4.7 TAG File Output Format Option 42-16
42.4.8 Keytag File Output Format Option 42-18
42.4.9 HARDCOPY Output Option 42-19
42.4.10 Primary/Secondary Sorting Considerations 42-20

42.4.11 SORT Work Files 42-20
42.5 Disk space requirements 42-21
42.6 LINK into SORT from programs 42-21
42.7 The Use of CHAIN with SORT 42-26
42.7.1 Defining a Chain File for SORT 42-26
42.7.2 Naming a repetitive SORT procedure 42-27
42.7.3 Using CHAIN to cause a merge 42-28
42.8 SORT Execution-Time Messages 42-28

43. SUR COMMAND 43-1
43.1 Purpose 43-1
43.2 Use 43-1
43.2.1 Establishing a "Current Subdirectory" 43-2
43.2.2 Creating a Subdirectory 43-2
43.2.3 Deleting a Subdirectory 43-2
43.2.4 Renaming a Subdirectory 43-2
43.2.5 Displaying Subdirectories 43-3
43.3 About Subdirectories 43-3
43.3.1 Creation of Subdirectories 43-4
43.3.2 Deletion of Subdirectories 43-4
43.3.3 Being "in a Subdirectory" 43-4
43.3.4 Scope of a File Name 43-5
43.3.5 About Subdirectory SYSTEM 43-5
43.3.6 Files vs. the User Being "in a Subdirectory" 43-6
43.3.7 Getting a File into a Subdirectory 43-6

44. UBOOT COMMAND 44-1
44.1 Purpose 44-1
44.2 Use 44-1
44.3 UBOOT System Load Operation 44-2

45. UTILITY/OVL 45-1

46. UTILITY/REL 46-1
46.1 Printer Drivers 46-1
46.1.1 Print Driver Routines 46-2
46.1.2 ASA Control Characters 46-3
46.2 SECINOUT Drivers 46-3
46.2.1 SECINOUT Driver Routines 46-4

47. UTILITY/SYS 47-1

48. SYSTEM DESCRIPTION 48-1
48.1 System Philosophy 48-1
48.2 System Structure 48-1

49. SYSTEM STRUCTURE 49-1
49.1 Disk Structure 49-1

49.1.1 Introduction 49-1
49.1.2 Disk Space Management: CAT and Lockout CAT 49-2
49.1.3 Files: HDI, Directory Mapping Bytes, Directory, R49-3
49.1.4 Sector Identification 49-4
49.1.5 Addressing Byte Structures 49-5
49.1.5.1 PDA - Physical Disk Address 49-5
49.1.5.2 RIB Address/Protection 49-5
49.1.5.3 Segment Descriptor - used in RIB to define a segment. 49-6
49.1.5.4 Physical File Number - used to access directory and HDI 49-6
49.2 Disk Data Formats 49-7
49.3 Memory Mapping 49-8
49.4 Memory Tables 49-9
49.4.1 Entry Point Tables 49-9
49.4.2 Logical File Table 49-9
49.5 Disk Overlays 49-11
49.6 The Command Interpreter 49-12

50. INTERRUPT HANDLING 50-1
50.1 Interrupt Mechanism 50-1
50.2 Interrupt Scheduler 50-1
50.3 Active Processes 50-3
50.4 Timing Considerations 50-4
50.5 DOS Interrupt Routines 50-5
50.5.1 SETI$ 50-5
50.5.2 CLRI$ 50-5
50.5.3 CS$ 50-6
50.5.4 TP$ 50-6
50.6 Programming Considerations 50-6
50.6.1 Background Code 50-6
50.6.2 Foreground Code 50-7

51. SYSTEM ROUTINES 51-1
51.1 Parameterization 51-1
51.2 Exit Conditions 51-2
51.3 Error Handling 51-2
51.4 Foreground Routines 51-2
51.4.1 CS$ - Change Process State 51-3
51.4.2 TP$ - Terminate Process 51-3
51.4.3 SETI$ - Initiate Foreground Process 51-3
51.4.4 CLRI$ - Terminate Foreground Process 51-4
51.5 Loader Routines 51-4
51.5.1 BOOT$ - Reload the Operating System 51-4
51.5.2 RUNX$ - Load and Run a File by Number 51-5
51.5.3 LOADX$ - Load a File by Number 51-5
51.5.4 INCHL - Increment the H and L Registers 51-5
51.5.5 DECHL - Decrement the H and L Registers 51-6

51.5.6 GETNCH - Get the Next Disk Buffer Byte 51-6
51.5.7 DR$ - Read a Sector into the Disk Buffer 51-7
51.5.8 DW$ - Write a Sector from the Disk Buffer 51-8
51.5.9 DSKWAT - Wait for Disk Ready 51-8
51.6 File Handling Routines 51-9
51.6.1 PREP$ - Open or Create a File 51-10
51.6.2 OPEN$ - Open an Existing File 51-10
51.6.3 LOAD$ - Load a File 51-11
51.6.4 RUN$ - Load and Run a File 51-11
51.6.5 CLOSE$ - Close a File 51-12
51.6.6 CHOP$ - Delete Space in a File 51-13
51.6.7 PROTE$ - Change the Protection on a File 51-14
51.6.8 POSIT$ - Position to a Record within a File 51-15
51.6.9 READ$ - Read a Record into the Buffer 51-15
51.6.10 WRITE$ - Write a Record from the Buffer 51-16
51.6.11 GET$ - Get the Next Buffer Character 51-16
51.6.12 GETR$ - Get an Indexed Buffer Character 51-17
51.6.13 PUT$ - Store into the Next Buffer Position 51-18
51.6.14 PUTR$ - Store into an Indexed Buffer Position 51-18
51.6.15 BSP$ - Backspace One Physical Sector 51-19
51.6.16 BLKTFR - Transfer a Block of Memory 51-19
51.6.17 TRAP$ - Set an Error Condition Trap 51-20
51.6.18 EXIT$ - Reload the Operating System 51-22
51.6.19 ERROR$ -- Reload the Operating System 51-23
51.6.20 WAIT$ -- DOS Wait-a-While "NOP" Routine 51-23
51.7 Keyboard and Display Routines 51-24
51.7.1 DEBUG$ - Enter the Debugging Tool 51-24
51.7.2 KEYIN$ - Obtain a Line from the Keyboard 51-27
51.7.3 DSPLY$ - Display a Line on the Screen 51-28

52. DOS FUNCTION FACILITY (DOSFNC) 52-1
52.1 FUNC1 - Retrieve Directory and C.A.T. Addresses 52-2
52.2 FUNC2 - Retrieve Directory Sector or Filename 52-5
52.3 FUNC3 - Retrieve RIB Information 52-7
52.4 FUNC4 - Retrieve DOS Configuration Information 52-9
52.5 FUNC5 - Request Access to System Tables 52-10
52.6 FUNC6 - Keyboard / Display Interface Routines Function52-11
52.7 FUNC7 - Test the Disk Buffer Memory 52-14
52.8 FUNC8 - Timed Pause 52-15
52.9 FUNC9 - Non-Sharable Resource Status Request 52-16
52.10 FUNC10 - Partition Information Function 52-18
52.11 FUNC11 RAM Screen Loader 52-19
52.12 FUNC12 - Enable Memory Resident Overlays 52-21
52.13 Overlay Loader (FUNC-13,14,15) 52-22
52.14 FUNC-13 Overlay Lookup By Name 52-24
52.15 FUNC-14 Load Absolute Library Member 52-25
52.16 FUNC-15 Relocatable Loader 52-26
52.17 FUNC-16 Disable Memory Resident Overlays 52-28

53. CASSETTE HANDLING ROUTINES 53-1
53.1 TPBOF$ - Position to the Beginning of a File 53-2
53.2 TPEOF$ - Position to the End of a File 53-2
53.3 TRW$ - Physically Rewind a Cassette 53-3
53.4 TBSP$ - Physically Backspace One 53-3
53.5 TWBLK$ - Write an Unformatted Block 53-3
53.6 TR$ - Read a Numeric CTOS Record 53-4
53.7 TREAD$ - TR$ and Wait for the Last Character 53-4
53.8 TW$ - Write a Numeric CTOS Record 53-5
53.9 TWRIT$ - TW$ and Wait for the Last Character 53-5
53.10 TFMR$ - Read the Next File Marker 53-6
53.11 TFMW$ - Write a File Marker Record 53-6
53.12 TTRAP$ - Set an Error Condition Trap 53-7
53.13 TWAIT$ - Wait for I/O Completion 53-8
53.14 TCHK$ - Get I/O Status 53-8

54. COMMAND INTERPRETER ROUTINES 54-1
54.1 CMDINT - Return & Scan MCR$ line 54-1
54.2 DOS$ - Return & Display Sign On 54-2
54.3 NXTCMD - Return and Dislay "READY" 54-2
54.4 CMDAGN - Return & Give Message 54-2
54.5 GETSYM - Get Next Symbol 54-3
54.6 GETCH - Get the Next Character 54-3
54.7 GETAEN - Get Auto-Execute Physical File Number 54-4
54.8 PUTAEN - Set or Clear a File to be Auto-Executed 54-4
54.9 GETLFB - Open the User-Specified Data File 54-4
54.10 PUTCHX - Store the Character in "A" 54-5
54.11 PUTCH - Alternate Version of PUTCHX 54-6
54.12 PUTNAM - Format a Filename from Directory 54-6
54.13 MOVSYM - Obtain the Symbol Scanned by GETSYM 54-7
54.14 GETDBA - Obtain Disk Controller Buffer Address 54-7
54.15 SCANFS - Scan Off File Specification 54-7
54.16 TCWAIT - Test controller memory & wait 54-8

55. USER SUPPORTED INPUT/OUTPUT 55-1

56. ERROR MESSAGES 56-1
56.1 System Error Messages 56-1
56.2 Utility Program Error Messages 56-3

57. ROUTINE ENTRY POINTS 57-1

58. PROCESSOR DEBUG 58-1
58.1 Introduction 58-1
58.2 Startup Procedure 58-1
58.3 Saving the Machine State 58-2
58.4 Display Format 58-2

58.5 The Command Interpreter 58-3
58.6 Command Syntax 58-3
58.7 Input Command List 58-4
58.8 DEBUG Command Summary 58-8
58.9 Extensions to Standard DEBUG 58-9

Appendix A. DOS.A AND DOS.E A-1
A.1 Planning for DOS.A/DOS.E A-1
A.1.1 DOS.A Physical Configuration A-1
A.1.2 DOS.E Physical Configuration A-2
A.2 Disk Drives A-2
A.3 Disk Media A-2
A.4 Loading and unloading Disk Cartridges A-2
A.5 Switches and Indicators A-3
A.6 Care and Handling of Disk Cartridges A-4
A.7 Care and Maintenance of the 9350 Drives A-5
A.8 Head Crashes A-5
A.8.1 Prevention of Head Crashes A-6
A.8.2 Recognition of a Head Crash A-6
A.8.3 What to Do if You Have a Head Crash A-7
A.9 Preparing Disk Packs for Use A-7
A.10 Disk Organization under DOS.A/DOS.E A-8
A.10.1 Logical Drive Mapping A-8
A.10.2 Size of a Logical Drive A-8
A.10.3 Cluster Mapping A-8
A.10.4 Segments under DOS.A A-9
A.10.5 Maximum File Size A-9
A.10.6 Cluster Allocation Table and Directory A-10
A.11 Internal DOS Parameterization A-11
A.11.1 Physical Disk Address Format A-11
A.11.2 Hardware Address Structure A-11

Appendix B. DOS.B B-1
B.1 Planning for DOS.B B-1
B.2 File Storage Capacity under DOS.B B-1
B.3 Disk Drives B-2
B.4 Disk Media B-2
B.5 Loading and unloading Disk Packs B-2
B.5.1 Models 9370-9373 B-2
B.5.2 Model 9374/9375 B-3
B.6 Switches and indicators B-4
B.6.1 Models 9370-9373 B-4
B.6.1.1 Memorex Drives B-4
B.6.1.2 "Telex" Drives B-5
B.6.1.3 Common Features B-5
B.6.2 Model 9374/9375 B-6
B.7 Care and Handling of Disk Packs B-7
B.8 Care and Maintenance of the 9370 Drives B-7

B.9 Head Crashes B-9
B.10 Preparing Disk Packs for Use B-9
B.11 Disk Organization under DOS.B B-10
B.11.1 Logical Drive Mapping B-10
B.11.2 Size of a Logical Drive B-11
B.11.3 Cluster Mapping B-11
B.11.4 Segments under DOS.B B-12
B.11.5 Maximum File Size B-12
B.11.6 Cluster Allocation Table and Directory B-13
B.12 Internal DOS Parameterization B-14
B.12.1 Physical Disk Address Format B-14
B.12.2 Hardware Address Structure B-14

Appendix C. INTRODUCTION TO DOS.C C-1
C.1 Planning for DOS.C C-1
C.2 Performance of DOS.C C-2
C.3 Disk Drives C-3
C.4 Disk Media C-3
C.5 Loading and Unloading Diskettes C-3
C.6 Drive Numbering and Switches C-5
C.7 Care and Handling of Diskettes C-5
C.8 Preparing Diskettes for Use C-6
C.9 Suggested Disk Organization Techniques C-7
C.10 Disk Organization under DOS.C C-8
C.10.1 Radius Spiraling and Sector Skewing C-8
C.10.2 Size of a Diskette C-10
C.10.3 Cluster Mapping C-10
C.10.4 Segments under DOS.C C-11
C.10.5 Maximum File Size C-11
C.10.6 Cluster Allocation Table and Directory C-12
C.11 Internal DOS Parameterization C-13
C.11.1 Physical Disk Address Format C-13

Appendix D. DOS.D D-1
D.1 Planning for DOS.D D-1
D.2 File Storage Capacity under DOS.D D-1
D.3 Disk Drives D-2
D.4 Disk Media D-2
D.5 Loading and Unloading Disk Packs D-2
D.6 Switches and Indicators D-4
D.7 Disk Organization under DOS.D D-5
D.7.1 Logical Drive Mapping D-5
D.7.2 Size of a Logical Drive D-6
D.7.2.1 Models 9370-9373 and 9390-9393 D-6
D.7.2.2 Models 9374/9375 D-7
D.7.3 Cluster Mapping D-7
D.7.4 Segments under DOS.D D-7
D.7.5 Maximum File Size D-8

D.7.6 Cluster Allocation Table and Directory D-8
D.8 Internal DOS Parameterization D-9
D.8.1 Physical Disk Address Format D-9

Appendix E. DOS.G - 1800 OPERATING SYSTEM E-1
E.1 CRT / Keyboard Interface Under DOS.G E-1
E.1.1 Screen Line Numbering E-2
E.1.2 Displaying on the Screen E-3
E.1.3 Inputting Data From the Keyboard E-3
E.1.4 Special CRT / Keyboard Features E-3
E.2 Diskette Organization Under DOS.G E-5
E.2.1 Loading and Unloading Diskettes E-5
E.2.2 Drive Numbering E-6
E.2.3 Care and Handling of Diskettes E-6
E.2.4 Preparing Diskettes for Use E-6
E.2.5 Sector Skewing E-6
E.2.6 Size of a Diskette E-7
E.2.7 Cluster Mapping E-7
E.2.8 Segments Under DOS.G E-7
E.2.9 Cluster Allocation Table and Directory E-8
E.3 Internal DOS Parameterization E-9
E.3.1 Physical Disk Address Format E-9

Appendix F. COMPARSION CHART FOR DOS'S F-1

Appendix G. DISK DATA FORMATS G-1
G.1 Disk Data Formats G-1
G.2 OBJECT File Format for Disk G-1
G.3 Relocatable Code Formats G-3
G.3.1 Directory G-4
G.3.2 Program Identification G-5
G.3.3 Object Text G-5
G.3.3.1 Memory Location G-6
G.3.3.2 Absolute Text G-6
G.3.3.3 Complex Relocatable References G-7
G.3.3.4 Simple Relocatable References G-8
G.3.4 External Definitions G-10
G.3.5 External and Forward References (4096 maximum) G-11
G.3.6 Transfer Address G-11
G.4 Format of Library Files G-12
G.4.1 Directory G-12
G.4.2 Members G-13
G.4.3 Library Type Chart G-14
G.5 DATABUS Code File Format G-14
G.6 DATAFORM Data File Format G-14
G.7 MULTIFORM File Format G-15
G.8 TEXT File Format G-16
G.9 ISI File Format G-18

G.10 SORT TAG File Format

CHAPTER 1. CHANGES FROM PRIOR VERSIONS

The following changes have been implemented in DOS since version 2.5 and its maintenance releases. All features and corrections from the version 2.5 DOS releases are included in the 2.6 releases unless otherwise noted below.

1.1 Utility Programs

1.1.1 BACKUP

Error detection and correction has been significantly improved. There is also a big performance improvement.

1.1.2 CAT

The CAT command will abort a search between drives if the keyboard key is depressed.

1.1.3 COPY

COPY now has an "E" option which, if set, will cause COPY to stop copying and chop the file when end of file is reached. If you press the keyboard key while COPY is clicking, COPY will now close and chop the file at that point.

1.1.4 DOSGEN

DOSGEN now writes on the volume being DOSGENed and then reads the data back.

1.1.5 DUMP

DUMP now has "P" and "Q" options for handling print files.

1.1.6 EDIT

Two new commands have been added to EDIT. The ":G*" command will display the line number of the pointed line. The ":T*" command will display the current tab settings.

1.1.7 INDEX

INDEX will now accept column numbers up to 32,767 for key and primary/secondary record specification.

1.1.8 MIN

MIN will now accept then "r" option to direct it to use the rear cassette deck. MIN also checks for the existance of cassette decks before executing.

1.1.9 PUTIPL

PUTIPL may now be run on a local volume while running under ARC.

1.1.10 PUTVOLID

PUTVOLID may now be run on a local volume while running under ARC.

1.1.11 REFORMAT

REFORMAT will now accept a record length specification of up to 65535.

1.1.12 SORT

SORT will now accept column numbers of up to 32767 for key and primary/secondary record selection. The SORT key trains are now sorted so an intermediate pass is not always required. SORT now requires at least 20k and a 5500 instruction set.

1.2 System Routines

There is no longer an arbitrary upper limit on file size. A file may be as large as will fit on one volume.

When running under ARC all DOS utilities permit access of up to 31 logical drives. However, only 16 of these may be local since in stand-alone mode DOS only permits accessing 16 drives. All 31 may be remote ARC volumes if you wish.

CHAPTER 2. INTRODUCTION

Datapoint Corporation's Disk Operating System (DOS) is a comprehensive system of facilities for sophisticated data management.

DOS provides the operator with a powerful set of system commands by which the operator can control data movement and processing from the system console. These commands allow the system operator to accomplish things which could be substantially more difficult on other computing systems. Sorting a large file, for instance, can generally be accomplished in a single command line. Despite the simplicity of operation, a wide range of features is provided.

To the programmer, DOS offers a set of facilities to simplify and generalize his task and file management problems. Concepts like dynamic disk space allocation allow programs to efficiently operate without regard to the amount of space required for the data files they are using. In addition, the disk file structure used by DOS allows for direct random access to data files. DOS also makes use of fully space-compressed text files.

These features, combined with the ability to support up to 200 million bytes of high-speed random access disk storage, provide a full range of data processing capabilities.

2.1 Hardware Support Required

The minimal configuration required to run DOS is a Datapoint processor (1100, 1800, 2200, 5500, or 6600 family), with a minimum 16K of memory, and one disk storage unit (9320, 9350, 9370, 9380, 9390, or 1840 series). For backup and support purposes, users with the Diskette 1100 series computer are required to have at least one system with more than one diskette drive. 1800 systems always use a minimum of 2 diskette drives. Configurations based on the other processors can operate with only a single disk drive unit in conjunction with the integral tape cassettes in most processors, but for backup and system support purposes a two-drive system is a recommended minimum.

2.2 Software Configurations Available

DOS is provided in several different models. Different models are used depending upon the type of processor and disk in use at an installation. Specific models are indicated by a letter after a period in the name of DOS. The following models of DOS are currently defined:

DOS.A -- Supports 9350 series disk drives on Datapoint 2200 and 5500/6600 family computers.

DOS.B -- Supports 9370 series disk drives on Datapoint 2200 and 5500/6600 family computers.

DOS.C -- Supports 9380 series disk drives on Datapoint 1100, 2200 and 5500/6600 family computers.

DOS.D -- Supports 9370 series and 9390 series disk drives on Datapoint 5500/6600 family computers and 9320 series disk drives on the 1800/3800 series computers. DOS.D is the host DOS for the Datapoint ARC system, and is one of the operating systems supporting the Datapoint Partition Supervisor systems.

DOS.E -- Supports 9350 series disk drives (with 16 buffer disk controller) on Datapoint 5500/6600 family computers. DOS.E also supports the Datapoint Partition Supervisor systems.

DOS.G -- Supports 1840 series disk drives on Datapoint 1800 family computers.

DOS.H -- Supports 1540 series disk drives on Datapoint 1500 family computers. (DOS.H is not described in this manual. See instead the DOS.H User's Guide, model code 50308.)

2.3 Program Compatibility

This manual describes the compatible set of facilities available to the DOS user within the Disk Operating System. Programs written in any of the supported higher level languages (DATASHARE, COBOL, BASIC, RPG II and others) will generally run unmodified on any of the DOS. Most programs written in assembler language will also run under any of the dot-series DOS, without reassembly.

Basically, in only a few cases will a program need to be changed when it is transferred from one DOS to another. The need for program modification will usually stem from one or more of the

following types of situations, which should be avoided whenever possible:

1) Programs which make assumptions regarding the size of files. For example, programs originally written for the 9350 series disks might assume that the size of the biggest possible file could be expressed as four ASCII digits. Under DOS.D, this assumption is invalid since files under DOS.D may be over 38,000 data sectors long.

2) Programs which make assumptions regarding the physical structuring of the data on the disks. For example, each DOS allocates space on the disk in segments of different sizes, and places its system tables in different locations on the disk.

3) Programs which generate or modify physical disk addresses themselves. Since the disks are each organized somewhat differently to take advantage of the particular characteristics of the specific type of drives involved, the physical disk addresses naturally vary among different DOS.

4) Programs which rely upon other characteristics of a DOS which are not documented in this manual. A possible situation would be where a programmer might look at the values in the registers following the return from a system routine and determine, for instance, that some routine always seemed to return with the value "1" in one of the registers. If he then constructs his program in such a manner that it will not function correctly if the "1" is not present upon return from the routine, then he is likely to find that his program will not work properly on a different DOS.

5) Programs which physically address the CRT/keyboard.

Only the first of the above situations will occur when using a high-level language. The others only occur in assembler language programs operating at the most detailed level of access to operating system routines. Programmers who require this level of detailed knowledge about the DOS will find the information specific to each DOS in the Appendix for the DOS they are using.

CHAPTER 3. OPERATOR COMMANDS

All Datapoint computers include, as a standard feature, an integral CRT display through which the internal computer communicates with the operator. The system console also includes a typewriter-style keyboard which the operator uses to communicate with the computer. The DOS is normally controlled by commands entered at this system console.

3.1 General Information

When DOS first becomes ready for commands, it displays a signon message on the CRT and says "READY". Upon completion of any job the DOS generally again displays "READY". Whenever the ready message is shown, the operator may key in a command, which will be displayed on the bottom line of the CRT as it is keyed in. While typing a command, the BACKSPACE key will erase one character for correction, and the CANCEL key will erase the entire line.

A command line specifies first what job is to be performed, then any disk files or special system directives, then options for the job. The command programs provided with DOS are described in this manual; the information that must be entered for each command is specified in the chapter about that command. A command line is always terminated with the ENTER key.

3.2 Command Line Syntax

In general, a command line is entered as:

<field>,<field>,<field>,<field>;[options]

Each <field> indicates a DOS file name specification (see the Disk Files chapter) or possibly a special field such as a subdirectory name. The first <field> on the line always specifies the program that will be run. Special attention must be given to the separators between fields on the command line. The most common separators are space and comma. For legibility the first two fields are usually separated by a space and subsequent fields are separated by a comma. A command then usually looks like:

SORT ACCTFILE,SRTFILE,:DR3;2-11

In this example the first field, the program to be executed, is "SORT". The second field is "ACCTFILE", the third is "SRTFILE", and the fourth is ":DR3". All of these fields provide information to the SORT program. A semi-colon (;) is a special separator which always separates <field> entries from [options]. In the above example the options field is "2-11". Slash (/) and colon (:) are special separators used within a file name.

Aside from the separators noted above, most special characters ($, ?, #, =, and so on) act as separators just like space or comma. In general, any character that is not a syntactically valid part of a file name will be interpreted as a field separator. The command example above could have been entered as:

```
SORT@ACCTFILE=SRTFILE$:DR3;2-11
```

Even / and : may be interpreted as field separators if not used as valid portions of a file name. Thus the command

```
COPY NAME/TXT/TEMP
```

has three fields: COPY, NAME/TXT, and TEMP. The use of special characters is not recommended since the resulting command line is very confusing for human interpretation.

3.3 Command Interpretation

As already noted, the first field on the command line specifies the program to be executed. For any command this first field must be given, any other fields may or may not be needed for a particular command. The command program must be a loadable object file or the program load will fail and the DOS will simply return to "READY" condition. If the program specified to be run cannot be found, the DOS displays the message "WHAT?" and waits for another command. If desired, the program name specification can be preceeded by an asterisk (*) or a colon (:), indicating the command is to be located in UTILITY/SYS in preference to a separate command file (See Command Interpreter section).

Fields on the command line are often order dependent. If a command is being used which accepts several fields, one of which is not wanted, skip that field by entering two separators with nothing between them.

```
SORT ACCTFILE,,:DR3;2-11
```

By using two commas, ":DR3" is recognized as the fourth field on the line, with the third field being null.

3.4 Documentation Conventions

When the command line is discussed in this manual, the first field is called the "command"; subsequent fields before the semi-colon are called "<filespec>" or some similiar term; characters following the semi-colon are called "options" or "parameters".

Prototype command lines will be shown in the form:

command [<filespec1>][,<filespec2>][,<spec3> . . .][;<options>]

Items enclosed in angle brackets ("<filespec>") represent a specification that will be entered on the actual command line. The angle brackets are not punctuation actually used on the command line. Square brackets also appear on the prototype lines but are not actually used as punctuation on the command line. Items enclosed in square brackets ("[;<options>]") represent optional fields that may be omitted or included as desired by the operator. Items on a prototype line that appear as capital letters represent the actual characters that must be entered. Items appearing as small letters represent the location for some different actual entry.

3.5 Program Signon Messages

When a command program begins execution it first displays a message identifying itself. If the command is specific to one single DOS, the signon message will also identify which DOS the command is designed to execute under. The main purpose of the signon message is to allow the operator to determine, in the event of some difficulty, whether a superseded version of the command program is in use.

CHAPTER 4. EQUIPMENT CARE

Computers, disk drives, printers, and other data processing equipment are delicate devices. They must be operated correctly and given a degree of care to continue to perform correctly. Datapoint prints "A Guide for Operating Datapoint Equipment", model code #60252, which gives detailed instructions on the operation of Datapoint equipment. It is recommended that any installation without trained computer operators obtain this manual.

4.1 Environment

Datapoint systems must be installed in an area with adequate air conditioning. Datapoint processors can stand a fairly wide range of temperatures, but disk drives should have a temperature range of 60 to 80 degrees F. (15.5 to 26.7 degrees C.). The temperature tolerance varies with the type of drive in use (diskette drives can stand a much wider temperature range) but the 60-80 degree range is safest. Humidity must be kept low enough to avoid condensation (below 80%) but high enough to avoid excessive static electricity problems.

The machine area must be reasonably clean and dust-free. Fanatic cleanliness is not necessary, but dust, cigarette ashes, spilled liquids, and so forth can seriously affect machine operation.

Processors and peripherals require fairly "clean" power to avoid erratic operation. Machine room power should be supplied from a completely separate transformer if possible. Be sure devices such as adding machines and power tools are not connected to the same power leads as computer equipment. The electric motors in these devices cause severe power line noise and will seriously affect machine operation. If necessary, isolation transformers are available to supply clean power for Datapoint equipment.

4.2 Processor

The only user maintenance on the processor is to dust and clean the cabinet, CRT screen, and keyboard occassionally and to clean the cassette decks. The cassette decks are especially sensitive to grime: dirty decks can cause read/write errors and can even destroy tapes. The decks are cleaned in the same way audio cassette decks are cleaned. Use tape head cleaner and a cotton swab to clean the tape heads and capstans; use a dry, lint-free cloth or swab to clean the pinch rollers. The cassette decks should be cleaned as necessary depending on use; normally every few months, as often as weekly if the decks get very heavy use.

Be sure the ventilation slots on the top and rear of the processor are never blocked, as impeded air flow will cause overheating.

4.3 Disks and Disk Drives

Be sure all operators know how to insert and remove disks in the disk drives. Disks must be stored properly in an environment similar to that for the equipment. Consult the appendices of this manual, or the Guide for Operating Datapoint Equipment, or the Datapoint Product Specifications (green sheets) for details on disk handling.

The disk drives must not be subjected to bumps or jolts or head misalignment can occur. Physical location of the drives must allow adequate air circulation for cooling purposes.

4.4 Other Peripherals

All peripherals should be dusted occasionally in keeping with the necessary environment cleanliness. Aside from printers, most Datapoint peripherals require practically no user maintenance. For any necessary care, consult the Guide for Operating Datapoint Equipment, the green sheets, or your Datapoint service representative.

Printer ribbons must be changed periodically to maintain print quality. Cloth ribbons left in use for too long can disintegrate, requiring a very messy clean-up of inky lint when the ribbon is finally changed, so check the ribbon occasionally. To avoid paper jams on printers, be sure the paper is aligned correctly when loaded, and be sure the paper has a free path into

the printer and as it emerges to the paper tray.

CHAPTER 5. DISK FILES

On all DOS-supported disks, information is stored in sectors, each of which contains 256 bytes of information. Sectors containing related information are organized in a single structured group called a file. All information on a disk will generally be organized in files, except for certain system tables.

5.1 File Names

From the console, files are identified by a NAME, EXTENSION, and LOGICAL DRIVE NUMBER. The NAME consists of up to eight alphanumeric characters (no special characters). Typical file names would include:

```
EDIT      PAYROLL
EMPLOYEE  JUL1075
23NOV76   X1
```

The EXTENSION must start with a letter and may be followed by up to two alphanumeric characters. If an extension is used in a file name, it is separated from the NAME by a slash (/). The extension further identifies the file and usually indicates the type of information contained in the file. A "TXT" extension means text and usually implies data or program source code. "ABS" implies program object code (absolute code) loadable by the system loader. "CMD" implies an object code file to be used as a command program from the system console. Other common extensions are: REL, ISI, DBC, OVn, SYS, PRT, BAS, and LEX.

The LOGICAL DRIVE NUMBER specifies on which logical drive the file is (or will be) located. The drive specification is identified by a leading colon (:) and has the form ":DRn" or ":Dn" or ":<volid>". When the ":DRn" or ":Dn" forms are used, the "n" is a number indicating the logical drive number as assigned at system installation. The ":<volid>" form allows logical volume identification, regardless of the physical drive on which the disk is located. "<volid>" is an eight character identifier placed on a disk by the PUTVOLID program.

The complete form of a file name is thus

```
NAME/EXTENSION:DRIVE
```

When a file name is entered as part of a command, all three parts of the name are not usually needed, though they can be specified. The presence or absence of a part of the file name is determined by the special separators "/" and ":". Syntactically correct file name entries are:

```
NAME/ABS:DR0    /ABS:DR1
NAME/REL        /TXT
NAME:D0         :D2
NAME            NAME:DOSD1
```

If a portion of the file name is not used, DOS applies default values; the default value used depends on the location of the name on the command line, and on the command in use.

The first field on any command line is the command program to be run. For this field, a NAME must be given, the default extension is CMD, and the default drive is any drive. (An "any drive" default usually means a search of all drives, starting with drive 0). If the command name is preceeded by an asterisk (*) or a colon (:), the default extension and all-drive search do not apply, as the leading character indicates the given name is to be located as a member of UTILITY/SYS (an "absolute library"), rather than searched for as a file.

The default values for file names given as parameters to a command are described separately for each command.

5.2 File Creation

Files are always created implicitly. That is, the operator never specifically instructs the system to create a given file. Any command that writes to an output file will write into an existing file or will automatically create a new file if necessary.

A file to be created will be created on the drive specified in its file name field or specified in default values applied to its name. When a file is being created on a specific drive, files with the same name and extension on other drives are unaffected. If no drive is specified in the name or by default, the file is created on any drive which has free space, the search for available space starting on drive 0. "Available space" means one free space in the drive's directory, in which to place the name of the new file, and at least one cluster of free space on the disk, in which to place the data the file will contain. (A "cluster" is the smallest unit of disk space that can be assigned to a file;

clusters are defined in the chapter on System Structure.)

5.3 File Deletion

Deletion of a file is performed explicitly by operator command, using the KILL command described later. No other programs delete an existing file, although procedures such as system generation and backup naturally destroy all files on the output disk.

5.4 File Protection

DOS files can be given three types of protection: write protection, delete protection, and no protection. If a file is write protected, it can be neither written upon nor deleted. If a file is delete protected it cannot be deleted, although it can be written over, effectively destroying any data previously in it. If a file has no protection it can of course be modified in any manner. The CHANGE command is used to set the protection of a file.

CHAPTER 6. SYSTEM GENERATION

Before a disk can be used with DOS it must first be prepared by writing onto it basic system tables. Also, a surface verification must be performed so any bad areas of the disk surface will not be used. On a new installation, the system utility programs must be placed onto the disk for use. All these operations constitute system generation.

6.1 Initial Generation

Datapoint distributes DOS in two forms: as a set of cassette tapes or as a completely generated disk. Users who receive the complete disks need not perform the cassette generation described below, as it has already been performed on their disk. Anyone requiring additional working disks should generate them as outlined in "Scratch Disk Preparation".

6.1.1 Formatting

Before a disk can be written or read on any drive, it must be appropriately formatted. Cartridge disks for use on Datapoint drives (9350 series) require no formatting because they use hardware formatting -- the sector formatting is inherent to the disk. Datapoint diskettes (9380 series) are formatted when received and do not require a special formatting process before they can be used.

Diskettes for use on the dual-density drives (1840 series) must be formatted when first used. The DOSGEN and BACKUP programs of DOS.G are able to format the diskettes in the necessary manner.

The mass storage disks (9370 or 9390 series) also require a special formatting process before they can be used. The first tape of the DOS generation cassettes for mass storage operating systems (DOS.B and DOS.D) is a formatting program. Simply insert the cassette in the rear cassette deck and depress RESTART/RUN (on the 2200 processors only RESTART need be depressed). The tape will rewind and then load the formatting program INITDISK. This program will ask for a specific physical (not logical) drive number containing the disk to be formatted. After receiving a reply, the program will ask if the operator is certain the drive number is correct and the disk in it is scratch, since formatting

destroys any information previously on the disk. Formatting will then proceed. When finished, the program will display a message indicating the pack is completely formatted.

For additional information on the formatting program, see the chapter on INITDISK.

6.1.2 Cassette System Generation

The first tape of the DOS generation cassettes (second tape, for DOS.B and DOS.D) is the actual generation cassette. To use this cassette load it into the rear cassette deck and depress RESTART/RUN (on 2200 processors only RESTART needs be depressed). The tape will rewind and then load the DOS generation program. Loading takes about a minute. When the program has loaded it will display a sign-on message and ask what logical drive is to be generated. The drive specified must be on-line with a ready disk in it.

Following drive selection the program will ask if a full generation is desired. To get a full DOS generation, answer Y; for a partial gen (useful only for upgrades from an older version DOS) answer N. Partial generation is described below. Following selection of full generation, the program will ask to be sure the disk in the selected drive is scratch, containing no valuable files that would be destroyed by generation.

After the verification question, the program performs a surface test on the cylinders used by DOS for its system tables and operating files. If this test fails, the disk is considered unusable and error messages will so indicate. After a short pause for the above test, the program will ask if any cylinders are to be locked out. The normal answer to this question is N, since locked-out cylinders cannot be used by DOS. If it is desired to lock out any cylinders for special use, consult the DOSGEN chapter for a description of cylinder lockout.

The next step in system generation is a quick surface verification of the entire disk surface. If an error is encountered, the program displays the cylinder number in which the error occurred, beeps, and flags the cylinder in the Lockout CAT so the DOS will not use it.

Following surface verification the basic system tables are built on disk and the system programs are loaded from the tape. Programs loaded are SYSTEM0/SYS - SYSTEM7/SYS, CAT/CMD, MIN/CMD, and UBOOT/CMD.

For initial generation of mass storage disks, be sure to repeat the above procedure the proper number of times to assure the entire physical volume is generated with all logical volumes. To save time, the first logical volume of a physical pack can be fully generated (finish loading utilities as described below), then the remaining logical volume(s) on the pack can be generated using the disk DOSGEN command described later.

After loading the system programs, system generation is complete except for loading utilites, and the new DOS is brought up ready for commands.

For 2200 and 5500 systems which need to be booted from a cassette boot loader, as soon as the system is ready (easy to tell since the message on the CRT is "READY") enter the command UBOOT to produce a boot tape for the DOS. UBOOT will ask for a blank tape in the front cassette deck and will then write and verify a boot block on that tape. It is wise to make at least two boot tapes at this time, since the boot tape is the only way to start up DOS. Any time it is necessary to start DOS (after the processor has been turned off, after loading a different set of disks, etc.) simply place the boot tape in the rear deck and depress RESTART/RUN (RESTART only on 2200) to boot the operating system.

To completely finish system generation, the system programs and utilities must be loaded. These files are contained on the second and third tapes of the system generation cassettes (third and fourth tapes for DOS.B and DOS.D). To load the commands simply place each cassette in turn into the front cassette deck and enter the command

```
MIN;AO:Dn
```

where n is the drive number being generated. When the files on these two tapes have been copied to disk, generation is finished.

The generation cassettes for DOS.C include a fourth tape of system commands, containing all the programs in UTILITY/SYS (see the appropriate chapter in this manual) as separate files. These files are provided as a convenience so that only desired programs can be placed on a system diskette, leaving free space on the diskette for other use.

6.2 Partial Generation

The DOS generation tape program has an option to perform a partial generation for purposes of upgrading an older version of DOS to the present version. To use partial generation load the gen tape and specify the drive to be generated. When the program asks if a full generation is desired answer N. The program will ask a couple of verification questions to be sure it should just replace the system and command files, and will then do so.

During partial generation the eight system files SYSTEM0/SYS through SYSTEM7/SYS are replaced by new files from the tape. The old utility programs must be deleted and new programs loaded from tape before partial generation is complete. If the disk being upgraded includes an existing UTILITY/SYS file, it may be necessary to use LIBSYS to upgrade the absolute library, rather than simply overstoring the old library with the new one. For more information, see the chapter on UTILITY/SYS.

When performing a partial generation on a DOS. 1.1, 2.1, or 2.2 disk, it will be necessary to replace the old MIN/CMD with the new command from the generation tape before the utilities tapes can be loaded. (The old MIN cannot recognize the file format of UTILITY/SYS.) The replacement operation must be performed before the partial generation from the DOSGEN tape is performed.

To replace MIN, load the generation cassette in the front deck and run MIN (the old command already on disk). MIN will identify the tape as "CTOS SYSTEM TAPE FORMAT" and will scan the tape to find the CTOS catalog. When the catalog is located, the files on the tape will be displayed and MIN will ask

LOAD B?

Skip the file named B by answering "N", skip CAT in the same manner, then answer "Y" to load MIN. The program will ask for a DOS file name; the name given should be "MIN/CMD". MIN will ask to be sure the existing command should be overwritten, answer "Y" to the OVERWRITE? question. Once MIN/CMD has been loaded, enter an asterisk to end the program when it asks if UBOOT should be loaded.

After MIN/CMD has been replaced, use the new MIN to load the utility tapes in the normal manner.

Following a partial generation, it is a good idea to BACKUP the upgraded disk with reorganization. The reorganization removes any fragmentation in system files and allows an operator to easily

delete undesired old files. Until the old command files have been deleted, be sure to enter a leading * on each command so as to use the new utilities from UTILITY/SYS.

Partial generation is not valid between some versions of DOS (notably Version 1 DOS.B and any newer version). Check with your Datapoint System Engineer before attempting an upgrade by partial generation.

6.3 UPGRADE/X

A disk-based upgrade facility is available in a file called UPGRADE/X, X being the letter specification of the DOS in use. UPGRADE is a standard text file to be used as a chain procedure by the command

```
CHAIN UPGRADE/X;OUTPUT=:Dn
```

where n is the drive number containing the disk to be upgraded.

The UPGRADE procedure copies the eight system files from the new version disk (which should be in drive zero) to the specified drive. SYSTEM7/SYS is copied by use of COPY SYSTEM7/SYS;7 to preserve the subdirectory structure on the old disk. After the system files are copied, old utilities on the output disk are deleted and new utilities are copied from the input disk. The program PUTIPL is then run to place the necessary IPL blocks on the output disk.

Since UPGRADE is a text file, it can be edited to modify the chain procedure followed, to adjust to special needs. Any modifications performed should be very carefully considered to assure a good upgrade. System conversions are a complex process and any errors can result in an unusable disk or lost data.

As with partial generation from cassette, use of UPGRADE is not valid for all possible versions of DOS. Check with your Datapoint System Engineer before using UPGRADE for a disk conversion.

6.4 Scratch Disk Generation

Any disk to be used in a DOS system must be generated to contain the necessary system tables and basic system files. Scratch disks or new system disks are best produced by use of the DOSGEN program described later in this manual. DOSGEN is a totally disk based program and performs much more quickly than cassette generation. If necessary or desired, the DOS generation cassette can be used to produce a new disk, as described above in Initial Generation.

6.5 Generation Cassettes and Emergencies

If all boot tapes at an installation are lost or destroyed, there is suddenly no way to access perfectly good disks. New boot tapes can be made by loading the DOS generation cassette in the rear deck and pressing RESTART/RUN, then holding down the KEYBOARD key while the tape loads. After about 30 seconds a READY message will appear on the screen from the CTOS (Cassette Tape Operating System), which has just been loaded. Enter the command "RUN B" and CTOS will load and run the program called "B", which is a cassette-compatible version of UBOOT, producing a new boot tape for the DOS.

The generation tapes also provide an excellent backup copy of all system utilities and of the system files themselves. The system files are on the DOS generation tape as files #21 through #30 (SYSTEM0/SYS through SYSTEM7/SYS respectively). The availability of such backups can be invaluable in event of massive data loss on system disks.

CHAPTER 7. ABTONOFF COMMAND

7.1 Purpose

The ABTONOFF command is used to manually modify the ABTIF bit in DOSFLAG (see the description of //ABTIF in the chapter on the CHAIN command.)

7.2 Use

The command line for ABTONOFF is:

ABTONOFF [<condition>]

Where <condition> is one of "ON" or "OFF", specifying the desired condition of the bit. The command will display the prior condition of the bit before modifying its status. If it is desired to just manually inspect the bit without modifying it, specify no <condition>.

CHAPTER 8. APP COMMAND

8.1 Purpose

The APP command appends two object files together creating a third. Object files are files containing absolute object code in a format that can be loaded by the DOS loader.

8.2 Use

APP <file spec>,[<file spec>],<file spec>

The APP command appends the second object file after the first and puts the result into the third file. Note that neither of the input files are disturbed. If extensions are not supplied, ABS is assumed. The first two files (if a second is specified) must exist. If the third file does not already exist, it will be created. The first file's transfer address is discarded and the new file is terminated by the transfer address of the second file. The transfer address of an object file is defined as the entry point of the program contained in the file.

Omitting the second file specification causes the first file to be copied into the third file. For example:

APP DOG,,CAT

will copy the file DOG/ABS into the file CAT/ABS.

The first and third file specifications are required. If either is omitted the message

NAME REQUIRED

will be displayed. The second and third file specifications must not be the same.

Because the APP command recognizes the actual end of the object module contained in a file, APPing an object file, similiar to the example above, is one technique for releasing excessive unused space at the end of an object file.

Another use of the APP command is to append patches to object files, since the object files being APPed may load at the same address as the object code in the original program. However, since the new code is at the end of the module, it is loaded over the old code.

CHAPTER 9. AUTO COMMAND

9.1 Purpose

The AUTO command sets a program to be automatically executed when the DOS is loaded. This auto-execute feature may be used to facilitate automated procedures for unattended operation (as described in the chapter on AUTOKEY), or for a variety of other purposes. AUTO can also simply change the program chosen for auto-execution. Auto-execution is cleared by use of the MANUAL command.

9.2 Use

AUTO is invoked by the command

AUTO [<filespec>]

<filespec> is the name of the file to be auto-executed and must be an executable object file on the booted drive of the system. The default extension for <filespec> is ABS.

Following the program signon message, a line will be displayed describing the prior status of auto-execution. If no program was set to auto-execute, no message will be displayed. If there was a program set for auto-execute, the message

AUTO WAS SET TO filename/ext (pfn).

will appear, where "filename/ext (pfn)" specifies the name and physical file number of the old auto-execute program. The optional <filespec> on the command line is required if there was not a previous program set for auto-execution. If the <filespec> is omitted when required the message

NAME REQ'D

will be displayed and the program will terminate.

If a <filespec> is given in the command line the message:

AUTO NOW SET TO filename/ext (pfn)

will be displayed (where pfn is the physical file number). When the named file is set for AUTO execution a check is made to see if the file is an object file and if the file is on the booted drive. If the specified file does not exist, the message:

NO SUCH NAME

will be displayed.

9.3 Operation of AUTOed Program

When a program is set to auto-execute, it will execute any time the DOS is initialized. The automatic execution can be suppressed by holding down the KEYBOARD key on the processor. When the KEYBOARD key is depressed, DOS enters the normal command interpreter ("READY" message) after initialization, regardless of any program set for auto-execution.

Sometimes when a disk will not boot, the problem is caused by an error in the auto-execute program. The program to be auto-executed is identified only by its physical file number, so if something has written over the program or if the program file has been deleted, the system will hang up when an attempt is made to execute the file. To bypass this kind of problem simply boot the DOS while holding down the KEYBOARD key (to suppress auto-execution) then execute the MANUAL command to clear auto-execution.

The AUTO command specifies only the program to be executed, providing no additional information. When the specified program is executed it is not given any command line information, since no command line is provided. This characteristic of auto-execution makes it impossible to use AUTO for programs requiring or even accepting parameters from the command line. Programs using overlay files with the same name and a different extension will not work when auto-executed because the name used to load the overlay is usually obtained from the command line information. Since almost all programs use command line information, very few programs can be executed directly from auto-execution.

To overcome the limitations of direct auto-execution, the command AUTOKEY makes it possible to extend the capabilities of AUTO to encompass practically all programs. AUTOKEY is described in a separate chapter later.

Automatic program execution is implemented by storing the physical file number (PFN) of the file to be automatically

executed. The PFN stored for this purpose is called the Auto-Execute physical file number, or AEN. Whenever the operating system is initialized by execution at the DOS$ entry point, the file represented by the AEN of the booted drive is executed. Normally the AEN is zero, representing no auto-execute program. AUTO may set the AEN to any other value, allowing automatic execution of any program, subject to the restrictions noted above.

CHAPTER 10. AUTOKEY COMMAND

10.1 Purpose

Many users allow their Datapoint computers to run in an unattended mode. This allows large data processing tasks, perhaps running via the DOS command chaining facility (see CHAIN), to be run during the evening hours when no operator is present. (An example might be the creation of several new index files for one or more large, ISAM-accessed data bases). However, the momentary power failures which data processing users are being forced to contend with during times of shortage, thunderstorms and the like can bring down any computer not having special, uninterruptible power supplies. When this happens to a computer running in unattended mode, the office staff will generally return the next morning to find their computer sitting idle and its work unfinished.

The Datapoint computers are all equipped with an automatic-restart facility which can be used to cause them to automatically resume their processing tasks following such an interruption. The purpose of the AUTOKEY (and AUTO) commands is to provide a software mechanism for users who wish to handle such unusual circumstances and provide for the restarting of a processing task.

10.2 Use

To specify a command line to be used during automatic system restart, simply enter:

AUTOKEY [<command line string>]

at the system console. If no <command line string> is entered, AUTOKEY will display the current autokey line if there is one and then ask if this line is to be changed. If "N" is answered, AUTOKEY simply returns to the DOS and the DOS "READY" message is displayed. If "Y" is answered, AUTOKEY requests the new command line to be configured and then returns to the DOS and "READY".

Alternatively, if the user wishes to simply specify a new command line to be configured regardless of the current setting of

the AUTOKEY command line, he can merely place the new command line string after the "AUTOKEY" that invokes the AUTOKEY command. This will cause AUTOKEY to simply display the old and new commmand lines and return to DOS.

10.3 The Hardware Auto-Restart Facility

There are a number of ways Datapoint processors handle auto-restart.

10.3.1 Processors with tape decks

There are two small tabs on the back edge (directly opposite from where the tape is visible) of each cassette tape. The leftmost of these (as you look at the top side of the cassette) is the write protect tab, which prevents writing on the topmost side of the tape. The right-hand tab is the auto-restart tab.

Users who frequently use both sides of cassettes will probably immediately notice that if one turns over the tape, the assignments of these two tabs switch around, the tab which had been write protect now being auto restart and vice versa. This in fact is precisely what happens.

If the auto-restart tab on the rear cassette is punched out (or slid to the side), then the computer will automatically re-boot, just like it does when RESTART/RUN is depressed, whenever the processor goes to STOP. Assuming that the rear cassette drive contains a DOS boot tape, this will cause DOS to come up and execute any program set for auto-execution.

10.3.2 Processors without tape decks

The Datapoint 1100 family of processors are provided with switch-selectable auto restart. The computer will either halt or automatically restart upon being stopped, depending upon the setting of an internal switch. This switch can be set by a Datapoint representative upon request.

Datapoint 1800 and 6000 users are provided with a firmware auto-restart. Thus if the machine ever halts (due to a power failure for example), on being started again it will attempt to load the operating system.

10.4 Automatic Program Execution Using AUTO

In order to provide a mechanism for programs to resume automatically following an interruption (such as a DATASHARE system, for instance, which might be running unattended) DOS has a comparable facility to enable a program to be automatically executed whenever DOS comes up. (Note that any loading and running the DOS, whether by an auto-restart, executing the RESTART procedure, or under program control, will activate this facility.)

The AUTO command is used to establish a program to receive control when DOS comes up. This setting can be cleared with the MANUAL command. For some applications, the AUTO and MANUAL commands are adequate to allow a programmed restart of a lengthy data processing task. However, some programs require parameters be specified on the command line, and these are obviously not present if no command line has been provided.

10.5 Auto-Restart Facilities Using AUTOKEY

AUTOKEY is simply a command program which can be AUTOed. The way in which it works is very simple. If it is run via the DOS auto-restart facility, AUTOKEY supplies a command line just as if the same one line were entered at the system console.

The command line supplied to AUTOKEY could do anything that can be specified in one command line to the DOS; DATASHARE could be brought up, a SORT invoked, a user's own special restart program started or even a CHAIN begun. AUTOKEY, when used with AUTO, MANUAL, and CHAIN can therefore provide a very powerful facility.

10.6 A Simple Example

As a simple example, assume that XYZ Company has several of their sales offices on-line to their home office DATASHARE system, which is running completely unattended. Lightning strikes a powerline outside of XYZ Company's home office, and power is cut off for 15 seconds. As soon as power is restored, their Datapoint 5500 computer re-boots its DOS (since the right-hand tab on the boot tape has been punched out) and warmstarts the DATASHARE system. One command sequence to accomplish this would look like the following:

```
AUTOKEY
DOS. VER n.n AUTOMATIC KEYIN COMMAND
NO AUTOKEY LINE CONFIGURED.
CHANGE THE AUTOKEY LINE? Y
ENTER NEW AUTOKEY LINE:
DS55500
READY
AUTO AUTOKEY/CMD
AUTO NOW SET TO AUTOKEY/CMD (nnn)
READY
```

An alternate form of the above would be the following:

```
AUTOKEY DS55500
DOS. VER n.n AUTOMATIC KEYIN COMMAND
NO AUTOKEY LINE CONFIGURED.
ENTER NEW AUTOKEY LINE:
DS55500  <--- (this is supplied automatically)
READY
AUTO AUTOKEY/CMD
AUTO NOW SET TO AUTOKEY/CMD (nnn)
READY
```

Once a program has been set for auto-execution, the only way one can bypass it is to hold down the KEYBOARD key while the DOS is coming up. This action bypasses the auto-executed program and enters the normal command interpreter. The user then can use the MANUAL command to clear the auto-execution option.

10.7 A More Complicated Example

The following example uses many of the features of other facilities in the Datapoint system besides simply AUTOKEY. Explaining all of these in detail is beyond the scope of this section. The intention here is just to demonstrate the sophistication possible using AUTOKEY in conjunction with the other facilities within the DOS.

Let's assume that XYZ Company is running an eight-port Datashare system. Each of the company's seven sales offices around the country has a Datapoint 1100 computer which is connected up to the home office Datashare system as a port. (The eighth port is used by the home office's secretary, Susie.) During the day, each of the seven sales offices makes inquiries of the central inventory, price, and model code files through a system of Datashare programs, and another Datashare program lets

them key orders into a file called "ORDERSn" where n is their port number. At the end of each business day, XYZ Company wants to process these orders. First they put the seven files all into one large file, sort it, and use a Datashare program to make corresponding entries into the master order file. The master order file is then reformatted and the index reconstructed. The final step is to create a second copy of the master order file onto magnetic tape, which will then be saved for backup purposes.

Since the operation just described is fairly lengthy, one of the programmers at XYZ Company decided to allow it to run unattended after everyone has gone home. They even set up Susie's MASTER program so that it automatically takes down the Datashare system and starts up the end-of-day processing one-half hour after the company's Los Angeles sales office (two time zones behind the Chicago main office) closes for the afternoon. When the daily processing is completed, Datashare is brought back up again so that it will be up by the time the first people start arriving at the New York sales office the next morning, an hour before the Chicago main office opens.

In the event of an unanticipated power failure, the system will recover and bring itself back up, resuming operations at the last checkpoint established by AUTOKEY. Notice that the system is also left in a state such that after the chain completes, Datashare will automatically restart in the event of any possible system failure. (NOTE: Datapoint 9350 disk systems using Diablo disk drives will initialize with hardware in "WRITE PROTECT" mode after power interruption.)

The following chain file ("OVERNITE/TXT") accomplishes the preceeding, assuming that subdirectory "SYSTEM" is used throughout the chain. The chain file could be modified easily to eliminate this assumption. However, the chain file can be made almost arbitrarily complicated; the point here is simply to show one of many possible techniques for handling unattended operations which wish to restart automatically in the case of some failure. Notice that the chain file might have to be modified depending on the particular version of DATASHARE an installation is using.

```
// IFS S1
//. FIRST SET UP FOR AUTO RESTART IF REQUIRED.
AUTOKEY CHAIN OVERNITE;S1
AUTO AUTOKEY/CMD
BUILD NULL;!
!
//. NEXT APPEND TOGETHER THE SEVEN FILES.
SAPP ORDERS1,ORDERS2,SCRATCH
SAPP SCRATCH,ORDERS3,SCRATCH
SAPP SCRATCH,ORDERS4,SCRATCH
SAPP SCRATCH,ORDERS5,SCRATCH
SAPP SCRATCH,ORDERS6,SCRATCH
SAPP SCRATCH,ORDERS7,SCRATCH
//. NOW SCRATCH CONTAINS THE DAILY FILES.
//. SET FIRST CHECKPOINT AT END OF PHASE 1
AUTOKEY CHAIN OVERNITE;S2
// XIF
// IFS S1,S2
//. PHASE TWO SORTS FILE "SCRATCH" INTO "ORDERDAY".
SORT SCRATCH,ORDERDAY;1-5
//. NEXT CHECKPOINT HAVING BUILT "ORDERDAY".
AUTOKEY CHAIN OVERNITE;S3
// XIF
// IFS S1,S2,S3
//. PHASE THREE PROCESSES THE FILE WITH A DS55500 PROGRAM.
BUILD CONFIG/CHN;!
DS55500;C
Y
Y
Y
N
Y
N
N
Y
N
  <----------null line (just hit enter)
!
CHAIN CONFIG/CHN
DS55500 PROCESS
   .
   .
   .
```

The program PROCESS/DBC ends with ROLLOUT "CHAIN NULL" to end the program and continue the chain.

```
   .
   .
   .
```

```
//. THE MASTER ORDER FILE "ORDERMAS" NOW IS UPDATED.
//. SET NEXT CHECKPOINT
AUTOKEY CHAIN OVERNITE;S4
// XIF
// IFS S1,S2,S3,S4
//. PHASE FOUR REFORMATS THE MASTER ORDER FILE.
REFORMAT ORDERMAS,SCRATCH:WORK2;R
//. "SCRATCH" NOW IS A REFORMATTED COPY OF "ORDERMAS".
AUTOKEY CHAIN OVERNITE;S5
// XIF
// IFS S1,S2,S3,S4,S5
//. PHASE FIVE COPIES "SCRATCH" BACK TO "ORDERMAS"
COPY SCRATCH:WORK2,ORDERMAS
//. "ORDERMAS" IS NOW READY FOR INDEXING.
AUTOKEY CHAIN OVERNITE;S6
// XIF
// IFS S1,S2,S3,S4,S5,S6
//. PHASE SIX RECREATES THE INDEX FOR "ORDERMAS"
INDEX ORDERMAS;1-16
//. THE INDEX HAS NOW BEEN REBUILT.
AUTOKEY CHAIN OVERNITE;S7
// XIF
// IFS S1,S2,S3,S4,S5,S6,S7
//. NOW DUMP MASTER FILE TO 9-TRACK MAGNETIC TAPE.
TAPE ORDERMAS/TXT,I/E
B
O
200x4
X
*
//. NOW THE BACKUP COPY OF "ORDERMAS" IS ON TAPE.
AUTOKEY CHAIN OVERNITE;S8
//XIF
//IFS S1,S2,S3,S4,S5,S6,S7,S8
BUILD CONFIG/CHN;!
DS55500;C
Y
N
N
N
N
N
Y
N
3600
3600
3600
3600
```

```
3600
3600
3600
3600
  <------null line (just hit enter)
!
CHAIN CONFIG/CHN
KILL CONFIG/CHN
Y
KILL NULL/TXT
Y
AUTOKEY DS55500
//. AND START UP DATASHARE FOR NEXT DAY.
DS55500
// XIF
```

10.8 Special Considerations

When building long chain files that allow for automatic restart, several considerations must be made. Among these are that a file must not be changed in such a way that the change cannot be repeated if the previous checkpoint is actually used. To accomplish this goal, frequently the file being updated must be copied out to a scratch file, and the scratch file then updated. Following the completion of the update is when another checkpoint would be taken; following that the next phase would copy the updated file back over the original. Note that a checkpoint (that is, resetting the AUTOKEY command line) would have to be before the creation of the dummy copy to be updated; putting a checkpoint between the creation of the copy to update and the actual updating process could result in the updating of a partially updated copy. A little thought when choosing places to update the AUTOKEY command line is called for to ensure that the chain may be resumed from any of them without incorrect results.

10.9 AUTOKEY and DATASHARE

Some users who make frequent use of the DATASHARE ROLLOUT feature will notice that AUTO-ing AUTOKEY with the AUTOKEY command line set to DS55500;R will mean that whenever any port rolls out to any program or chain of programs, Datashare is automatically brought back up when that program or chain of programs finishes, regardless of whether or not DS55500;R was included at the end of the port's chain file.

CHAPTER 11. BACKUP COMMAND

11.1 Purpose

The BACKUP command provides for making copies of DOS disks. The user can make either an exact mirror image copy of the input disk or can select reorganization, which will group files by extension and file name, remove unnecessary segmentation and allow deletion of unnecessary files. Reorganization also allows copying of DOS disks onto disks with locked out cylinders that differ from those on the input disk. Some special considerations apply for specific disk configurations.

11.2 Use

A disk backup is initiated by the operator entering the following command:

BACKUP <input drive>,<output drive>[;options]

Input drive and output drive are specified as :DRn, or :Dn, or :<volid>. If the drive selected as the input drive is not protected (in "READ ONLY" mode), a message:

PLEASE PROTECT YOUR INPUT DISK
OR, TAP "DISPLAY" KEY TO CONTINUE

will be displayed. At this point protecting the input drive or holding down the DISPLAY key will cause the following message:

DRIVE n SCRATCH?

If the disk on drive n is scratch enter a "Y". Any other reply will cause the program to return to DOS. If you do reply "Y", the program will display the message:

ARE YOU SURE?

If you are absolutely sure that you want to write over the output disk, type "Y" again and press the enter key. Any other reply will cause the program to return to DOS. If the output disk has not been DOSGENed or the DOS file structure on it has been

damaged, the message:

DOSGEN YOUR DISK FIRST

will appear and control returns to DOS. If the output disk has been DOSGENed and seems in reasonable shape, the following message is displayed:

FILE REORGANIZATION?

If different cylinders are locked out on the input and output disks (if the disks' lockout CATs do not match), a mirror image BACKUP is not possible so the "FILE REORGANIZATION?" question is bypassed. Instead, a message appears specifying that reorganization is required and BACKUP with reorganization proceeds as described below.

If you wish to reorganize the files being transferred to the output disk, enter a "Y" in response to the reorganization question. In this case, see the section on reorganizing files for further instructions.

If you do not wish to reorganize your files and desire a mirror image copy of your input disk, enter an "N" in response to the reorganization question.

11.2.1 Options

DOS.C BACKUP is capable of "eliminating" the SYSTEM files on the output drive during reorganization. This saves 60 sectors on a DOS.C diskette system. The SYSTEM files are not really eliminated, however, they only take the minimum amount of space possible (3 sectors each on DOS.C). To eliminate the SYSTEM files, use a ";N" for [;option] on the command line and use BACKUP with reorganization.

The output diskette generated in such manner is not capable of booting or executing DOS by itself, however, if it becomes necessary to put the system files and the contents of that diskette onto a new diskette, use the ";S" option. The SYSTEM files from the booted drive will be copied onto the output diskette, along with the files from the input drive. This operation requires a three drive system.

11.3 Mirror Image Copy

If you have typed "N" in response to the file reorganization question, the program will ask the question:

DO YOU WANT THE OUTPUT COPY VERIFIED?

The output copy should always be verified, so answer this question 'Y'. Answering 'N' will result in a somewhat faster backup operation, but there will be no testing for parity errors on the output disk.

The program then asks:

DO YOU WANT TO COPY UNALLOCATED CLUSTERS?

Type "Y" and press the enter key if you want all data on the disk copied regardless of whether or not it is in an area allocated by DOS. This option is preferred in cases where you suspect that your DOS files may be partially destroyed or the output disk has never been fully initialized with data. Also use this mirror image copy if you have the 9374 disk system and one of the drive's heads gets misaligned. Backup will use the offset feature to try and retrieve your data. If BACKUP uses the track offset it will slow the program down but it could save your data.

Type "N" and press the enter key if you wish to copy your disk as quickly as possible without copying unused areas of the input disk. "Y" and "N" are the only replies allowed.

A mirror image backup makes the output disk a complete image of the input disk. Following mirror image backup the volid of the output disk is the same as the volid of the input disk, since even the volid sector was copied during the backup. Mirror image backup requires that the output drive have the same cylinders locked out as the input drive.

11.4 Reorganizing Files

If you have typed "Y" in response to the file reorganization question, the program will copy the System files, sort the Directory names, and allow the operator to delete files before copying the files to the disk copy.

Backup with reoganization to the booted drive is not possible.

11.4.1 Copying DOS to Output Disk

Various program status messages will appear during the copying of DOS. System tables are initialized and then the SYSTEMn/SYS files are copied to the output disk. The system tables themselves are not copied from the input disk to the output disk, as is done in a mirror image backup. Following backup with reorganization the volid of the output disk is unchanged from what it was before the backup.

11.4.2 Deleting Named Files

When all directory names have been sorted into file extension followed by file name sequence the following question will be displayed:

DELETE ANY FILES DURING REORGANIZATION?

Type "N" and press the enter key if all files are to be copied. Type "Y" and press the enter key if you wish to delete any files. If you reply "Y" a message asking which files are NOT to be copied will appear. The lower screen will be filled by a numbered list of files for you to choose from. Type the number or range of numbers (nn or nn-nn) found next to names of individual files you wish deleted. Type "ALL" and press the enter key if you wish to delete all of the files in the list. The files selected for deletion will be erased from the list. When all desired deletions have been made from a list, type "." and press the enter key to advance to the next list of file names.

When all file name lists have been examined, the program will advance to the copy named files phase.

11.4.3 Copying Named Files

Files with names in the system directory are copied in alphanumeric file extension, file name sequence. The name of each file is displayed as it is copied. All files are written as close together as possible with a minimum of segmentation.

11.5 Use of KEYBOARD and DISPLAY Keys

The KEYBOARD and DISPLAY keys may be pressed any time messages are being displayed. Depressing the DISPLAY key will hold the current display until the key is released. Depressing the KEYBOARD key will cause the program to terminate and return to DOS.

11.6 Error Messages

During the execution of BACKUP the following error messages may appear:

```
*** PLEASE PROTECT YOUR INPUT DISK ***
*** OR, TAP "DISPLAY" KEY TO CONTINUE***
```

Action: Write-disable the input drive.

INVALID DRIVE SPECIFICATION!

Action: Retype the BACKUP command with correct <input-drive> and <output-drive> specification.

ILLEGAL OUTPUT DRIVE!

Action: <input-drive> and <output-drive> have been specified as the same drive! Retype BACKUP command with correct specification.

BAD CLUSTER ALLOC TABLE!

Action: A bad Cluster Allocation Table has been detected on the input disk. The Cluster Allocation Table may be able to be fixed using the DSKCHECK command.

CYLINDER 0 OF BACKUP DISK IS UNUSABLE!

Action: Your scratch disk cannot be used for a system disk due to surface defects in cylinder 0. Use another output disk and start over.

SYSTEMn/SYS IS MISSING!

Action: Your DOS disk cannot be reorganized due to a missing system file. Catalog the missing system file on your input disk and start over.

PARITY ERROR ON DRIVE nn PDA: nnnn, nnnn

Action: An irrecoverable parity error has been detected on drive n during mirror image BACKUP operation. The address is shown for each error. If drive n is your output disk, DOSGEN must be rerun to lockout the bad addresses or use a different scratch disk for mirror image copy. If drive n is your input disk, new parity will be computed and the record will be copied. Note the error address and check for errors when copy is complete.

PARITY ERROR ON READ. LRN: nnnnn

PARITY ERROR ON WRITE. LRN: nnnnn

Action: An irrecoverable parity error has been detected during BACKUP with reorganization. The LRN shown in the message is the decimal system LRN at which the error occurred. The read error occurs only on the input disk; the write error occurs only on the output disk. Corrective action is the same as described above for parity errors during mirror image copy.

FORWARD OFFSET TRACK BEING USED

REVERSE OFFSET TRACK BEING USED

Action: On a 9374 disk system a parity error has been detected on the input drive and offset tracking is being used to try to recover the data. There will be 10 attempts on both sides of the track.

11.7 Reorganizing Files for Faster Processing

After a DOS disk has been used for awhile, the file structure becomes fragmented and related files become scattered. The more the disk is used the more total system performance is degraded due to increased disk access time. System degradation is especially noticeable when DATASHARE is being used. File reorganization using the BACKUP program is one way to clean up DOS disks and improve their efficiency.

BACKUP reorganization improves system efficiency by making the following changes:

- File segments are consolidated
- Files are packed more closely together
- Related files are clustered together

. Unused trash files are removed (optionally)

. Files are rewritten reducing marginal parity errors

11.8 BACKUP with CHAIN

Because BACKUP requires that its input drives be write protected, does not abort if parity errors occur during the backup, and may ask different questions depending upon the condition of the input and output disks, BACKUP generally should not be invoked from a CHAIN. Since the BACKUP operation is so critical to the protection of important files, an operator should monitor the entire backup operation.

11.9 Clicks during Copying

A click occurs each time an unused sector is copied (reorganization mode only). A file which, when copied, results in a lot of clicks (more than a dozen, perhaps) can probably be reduced in size, without any data loss, by using APP, SAPP or COPY;E as appropriate.

11.10 Special Considerations for BACKUP

When using BACKUP on the 9320, 9370 or 9390 disk packs, it is important to remember that each disk is more than one logical drive. Since BACKUP deals with logical drives, BACKUP must be run several times, once from each logical drive, to backup an entire physical disk.

With the 9374 and 9354 disk drives, it is important to remember that the drive contains a fixed platter that is a separate logical drive. BACKUP between the fixed and removable platters is possible.

CHAPTER 12. BLOKEDIT COMMAND

12.1 Purpose

The BLOKEDIT command provides for DOS text file manipulation. The command copies lines of text from any DOS text file(s) to create a new text file.

The BLOKEDIT command is useful for such things as:

New program source file generation by copying routines from existing program source files;

Existing program source file re-arranging by copying the lines of source-code into a new sequence (into a new source file).

Re-arranging lines or paragraphs of a SCRIBE file into a new file.

In this Chapter, the following terms apply:

Text file means a DOS text file as defined in the REFORMAT chapter.

Line means one line of a text file as displayed by the DOS LIST program.

12.2 Use

The syntax for the BLOKEDIT command line is as follows:

BLOKEDIT [<file spec>],<file spec>[;option]

The first file specification refers to the command file, if not specified the commands will be entered via the keyboard. The second file specification names the new (output) file. If no extension is supplied with the first file specification, TXT is assumed. If no extension is supplied with the second file specification, the extension given or assumed for the first file is used. If no drive is given for the first file, all drives are

searched. If no drive is given for the second file, the drive given or assumed for the first file is used. If no drive was specified for the first file specification then both files are opened on the first available drive. The specified output file must not exist on any drive on line unless the "O" option is used, in which case the file is overwritten.

12.3 File Descriptions

BLOKEDIT deals only with text files. For any given application there will be one text file called the COMMAND FILE which will hold the controlling commands for BLOKEDIT. Optionally the controlling commands may be entered directly to BLOKEDIT via the keyboard by omitting the command file parameter. There will be one or more text files called SOURCE FILES from which lines of text will be copied. And there will be one text file called the NEW FILE which will be the desired end result for the application.

12.3.1 Command Statement Lines

The command statements are the controlling factor for a BLOKEDIT execution. The command statements specify which source files will be used and which lines of text will be copied from them. If the command statements are to be read from a command file it must be generated by the DOS. EDIT command, or DOS. BUILD command, etc., before BLOKEDIT can be used.

There are three kinds of statement lines that are meaningful to BLOKEDIT: COMMENT lines, COMMAND lines, and QUOTED lines.

A COMMENT line is a line which has a first character of period.

This is an example of COMMENT LINES:

```
.
. THESE THREE LINES ARE COMMENT LINES.
.
```

As in program source files, a comment line may have explanatory notes or nothing at all following the period.

A COMMAND LINE is a line which has a SOURCE FILE NAME and/or source file LINE NUMBERS, or begins with a double quote symbol (").

The following are some example command lines:

```
FILENAME/EXT:DRO          NAME THE SOURCE FILE
1-100                     COPY LINES 1 THRU 100
350-377                   COPY LINES 350 THRU 377
150/TXT                   NAME THE SOURCE FILE
```

A command line must have a first character of an upper-case alphabetic character, or a digit, or a double quote symbol.

A command line that begins with an upper-case alphabetic character indicates that a new SOURCE FILE is being named. A new source file can be named only by putting the name of the file at the very beginning of the command line. Optionally, the extension and/or drive number for the file may be included with the source file name. If the source file name begins with a digit the file extension must be given.

A command line that begins with a digit indicates that the command line will have one or more numbers, which are the numbers of the lines to be copied from the source file previously specified into the new file.

A command line that begins with a double quote symbol indicates the beginning/ending of QUOTED LINES. The only information used by BLOKEDIT in a command line that begins with a (") is the (") itself, therefore the rest of the line can be used for comments.

A QUOTED LINE is a line between a pair of command lines which begin with a double quote symbol.

This is an example of QUOTED LINES:

```
" THIS IS THE BEGINNING OF QUOTED LINES COMMAND LINE.
INCMNT  HL    COUNT           POINT TO COUNTER
        LAM                   LOAD TO "A" REGISTER
        AD    1               INCREMENT BY 1
        LMA                   RESTORE TO MEMORY
" THIS IS THE ENDING OF QUOTED LINES COMMAND LINE.
```

There may be more than one quoted line between the command lines that begin with ("). A quoted line will be copied directly from the command file or keyboard to the new file. Quoted lines enable a BLOKEDIT user to include original lines of text in a new file along with lines copied from source files.

12.3.2 Source File

The SOURCE FILE is a text file from which lines will be copied. Source files are named in the command lines for a BLOKEDIT application, and the lines to be copied from the source file will also be specified in the command lines. It will be useful to have a listing of a source file with line numbers, as produced by the LIST command, when creating the command statement lines for a BLOKEDIT application.

12.3.3 New File

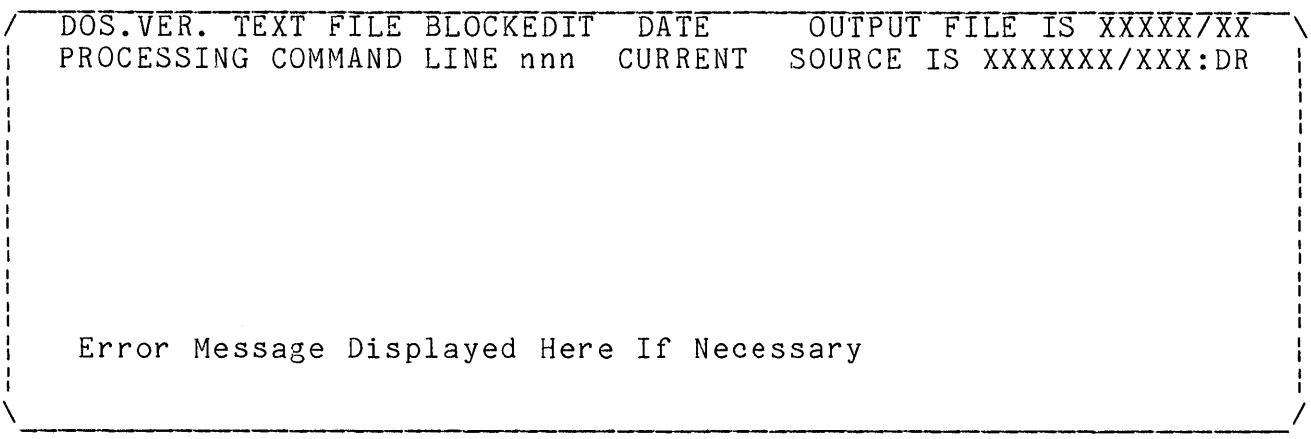
The NEW FILE is a text file produced by the BLOKEDIT command. The new file is named at BLOKEDIT execution time by the second file specification entered on the command line.

12.4 Messages

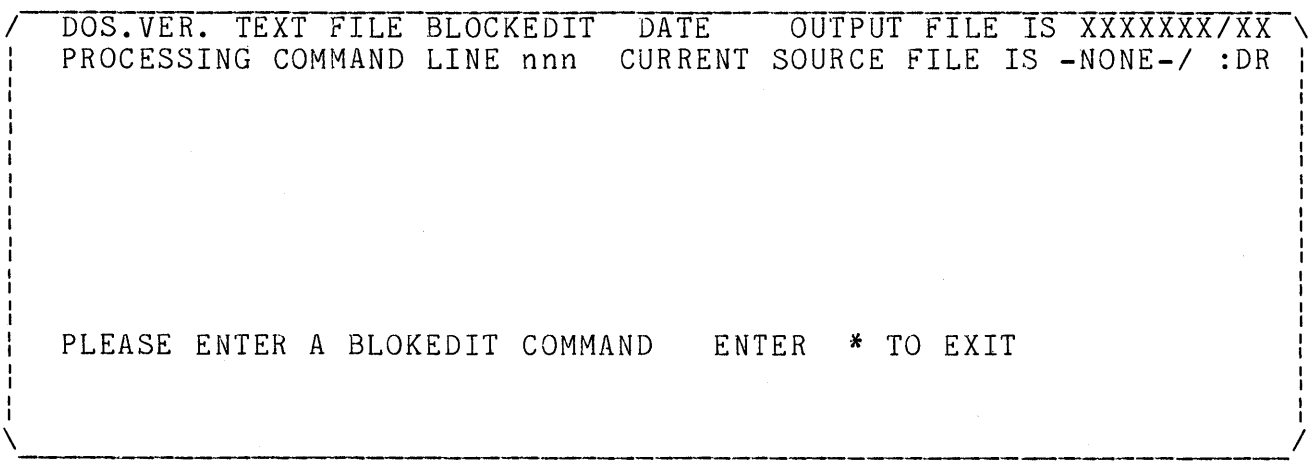
This section describes the operator messages that BLOKEDIT may display on the CRT screen during execution. Some of the messages are monitor messages to keep the operator informed of the progress of the program, while other messages are error messages. If the keyboard was selected as input to BLOKEDIT, the user will be prompted by the "Please enter BLOKEDIT command Enter * to exit." message when input is required. The character * will terminate BLOKEDIT and return to DOS.

The general format of the CRT display screen varies depending on the source of the BLOKEDIT command statements.

If the command statements are being read from a command file the format of the display is:



If the command statements are being entered via the CRT keyboard, the format is:



As BLOKEDIT commands are entered on the bottom line, previous lines are rolled up the screen.

12.4.1 Informative Messages

PROCESSING COMMAND LINE .. CURRENT SOURCE FILE IS ../..:DR.

This message is the BLOKEDIT monitor message. This message is displayed while BLOKEDIT is writing lines of text to the new file. The monitor message displays the command file line number currently being processed and the name, extension, and drive number of the last named source file.

SOURCE FILE WENT TO E.O.F.

This message is displayed if the source file from which lines were being copied ended before the specified lines were finished.

BLOKEDIT TRANSFER COMPLETE
OUTPUT FILE WAS name LINE COUNT WAS nnn

This message is displayed when all of the command file lines have been executed. The number of lines in the new file is displayed following the second line.

12.4.2 Fatal Errors

If BLOKEDIT detects a fatal error in the command statement line the monitor message is rolled up the screen, an appropriate error message is displayed, and the program aborts.

NEW FILE NAME REQUIRED

This message is displayed if the operator did not name a new file when the BLOKEDIT command was called.

COMMAND FILE DRIVE INVALID

This message is displayed if the operator specified for the command file a drive number that is invalid.

NEW FILE DRIVE INVALID

This message is displayed if the operator specified for the new file a drive number that is invalid.

COMMAND AND NEW FILE NAMES MUST NOT BE IDENTICAL

This message is displayed if the operator specified command file and new file names the same and the extension and the drives

for the files were specified or assumed to be the same. Default values of extensions and drives are described in an earlier paragraph.

COMMAND FILE NOT FOUND

This message is displayed if the command file name was not found on the drive(s) specified or assumed.

NEW FILE NAME IN USE
USE 'O' OPTION ON COMMAND LINE TO OVER-WRITE EXISTING OUTPUT FILE.

This message is displayed if the specified output file was found on the drive(s) specified or assumed. BLOKEDIT will not write into an existing file if commands are being read from a command file. If commands are being entered to BLOKEDIT via the KEYBOARD, the operator is given the option to overwrite the existing file:

***NEW FILE NAME IN USE, OVERWRITE IT? ANSWER WITH A YES OR NO

If the operator answers Yes (Y) the file is overwritten.
If the reply is No (N) BLOKEDIT returns control to DOS.

BAD FILE SPECIFICATION

This message is displayed if the first character of a command file line, other than a quoted line, is an upper-case alpha character but the DOS file specification was not recognizeable.

12.4.3 Selectively Fatal Errors

These errors are fatal when BLOKEDIT is reading a command file, and informative when commands are being entered via the keyboard.

SOURCE FILE NOT FOUND

This message is displayed if the source file specified could not be found. It is probably either misspelled or in a different subdirectory.

BAD LINE NUMBER SPECIFICATION

This message is displayed if a command file line other than a quoted line began with a digit but contained an unrecognizable line number specification.

Here are some examples of valid line numbers:

4	A single digit is acceptable.
999999	A line number may have up to six digits.
100-364	First and last line to be selected are separated by a dash.
34,55-78,100-147	Commas separate line specifications.

Here are some examples of invalid line numbers:

1A	Only "-", ",", or space after a digit, unless the line is a source file name beginning with a digit. If it is, an extension must be given.
1234567	Number has more than six digits.
17-34-77	Only two numbers separated by "-".

LINE NUMBER ZERO IS NOT VALID

This message is displayed if a line number of zero is specifed in a command line. It is ignored if entered via the keyboard.

START LINE NO. > END LINE NO

This message is displayed if the first number of a line number pair is larger than the second number of the pair, as in: 235-176. It is ignored if entered via the keyboard.

BAD DATA IN SOURCE FILE LINE nnn *

This message is displayed if BLOKEDIT discovers non-ASCII characters in a source file. The line number will be displayed following the message. If commands are being entered via the keyboard the source file is reselected, and next command is requested.

NO VALID SOURCE FILE FOR TRANSFER

This message is displayed if BLOKEDIT discovers line numbers to be transfered from a source file when there is no open source file.

FORMAT OR RANGE ERROR ON SOURCE FILE

This message is displayed if DOS discovers a file which can not be read. If commands are being entered via the keyboard the source file will be de-selected, and next command requested.

CHAPTER 13. BOOT

13.1 Purpose

The Alternate Drive Boot program provides a facility allowing a Datapoint disk user to boot to any drive in his system. This can be helpful with single drive systems using multiple logical drives per physical drive when it is necessary to DOSGEN or to BACKUP a pack with reorganization. If it is desired to DOSGEN disks on a single physical drive system, there is a peculiar problem: If DOS is running on drive 0, you cannot remove the disk in drive 0 and replace it with a scratch disk and continue with DOSGEN. This program lets you boot from drive 1 so that the DOSGEN program can be executed on drive 0. It is assumed that DOSGEN, CHAIN, COPY and PUTIPL are on drive 1.

13.2 Use

To boot DOS from a different drive than the currently "booted" drive, enter:

```
BOOT [:<drv>][;<new DOS command line>]
```

Where the new "boot" drive is specified by <drv> and the new DOS command line following the semi-colon will be executed when the boot process completes on the new drive. If you are not sure where the currently booted drive is, enter:

```
BOOT
```

This will display the location of the booted drive on the screen, and ask for a new boot drive.

13.3 Messages

WRONG DOS!!!
This program only operates on DOS version 2.5 or higher.

CURRENT "BOOTED" DRIVE IS :DRn.
This is a display of the currently "booted" drive.

THAT DRIVE IS OFF-LINE.
You requested booting from a drive that is not currently available.

NOW BOOTING DOS FROM :DRn.
DOS will be booted from the new drive.

ENTER NEW "BOOT" DRIVE NUMBER (0-nn).
If a drive number was not specified on the command line, you may enter a drive number here.

ENTER A DOS COMMAND LINE, OR TAP "ENTER".
When DOS is booted, it is possible to execute a command line (like CHAIN, for example) as soon as the booting process is done. If a DOS command line and a drive specification were omitted when this program was entered, you may now enter any valid DOS command line.

CHAPTER 14. BUILD COMMAND

14.1 Purpose

BUILD provides an alternative means to create a text file without having to use the standard DOS editor. BUILD is useful for rapid generation of very short text files, such as two and three line CHAIN files. Also, BUILD is usable from within a CHAIN.

14.2 Use

The BUILD command is invoked by entering the command line:

BUILD <file spec>[;<end character>]

The <file spec> defines the output file. This output file specification is always required. If the named file does not exist, it is created. The default extension is /TXT.

The <end character> is optional. If no end character is specified on the command line, BUILD terminates upon receiving a null input line (a null input line is a line consisting of only an ENTER; a blank line is not a null line).

BUILD accepts input lines from the keyboard and writes each one to the output file. When BUILD is ready to accept an input line it displays a colon (:) as a prompting character. Each input line BUILD receives is tested for the presence of the specified end character, if any, as the first character entered. If the end character is present as the only character of the entered line, the end line is discarded (it is not written to the output file), and an end of file mark is written to the output file and the output file closed by returning to DOS.

Entering an end character followed by a string will pass the string to the output line without the end character and will not terminate BUILD. This action allows entering CHAIN commands into a chain file being written by BUILD from within an active CHAIN.

14.3 A Simple Example

Suppose that the operator wishes to construct a simple CHAIN file to establish a program to be auto-executed, so that the auto-execute request can be accomplished later with a single command line entered at the keyboard. All that is required is to enter at the system console:

```
BUILD <chain file spec>;^
AUTOKEY <program name>
AUTO AUTOKEY/CMD
^
```

Upon receiving the "^" input line, BUILD closes the output file and terminates. Note that in the two places where the "^" appears, any enterable character could have been used. (This allows nesting calls to BUILD, which can be very useful in the BUILDing of chain files). After the BUILD command is finished, the output file named on the BUILD command line contains the following two lines:

```
AUTOKEY <program name>
AUTO AUTOKEY/CMD
```

It is also possible, through BUILD nesting, to create chain files which during execution of the chain construct other chain files and execute them automatically upon completion of the first chain (since any statement of a chain file is allowed to be a CHAIN command).

The chapter on the CHAIN command contains further examples of the use of BUILD from within a CHAIN procedure.

14.4 KEYBOARD and DISPLAY Keys

The KEYBOARD and DISPLAY keys may be pressed any time messages are being displayed. The keys will be effective just prior to the display of the prompting ":". Depressing the DISPLAY key will hold the current display until the key is released. Depressing the KEYBOARD key will cause the program to terminate and return to DOS.

CHAPTER 15. CAT COMMAND

15.1 Purpose

The CAT command selectively displays filenames in the DOS directory or in a library directory. One may choose to display all cataloged filenames on all drives online, or specific filenames on specific drives.

15.2 Use

The CAT command is invoked by entering the command line:

CAT [<name>][/<ext>][:<drv>][*][,L]

where: <name> specifies the filename or a portion of the filename, <ext> specifies the extension or a portion of the extension, <drv> specifies the logical disk drive, an asterisk indicates the named file is a library, and L specifies list only those files in the current subdirectory.

The default handling of the various input fields is any file name, any extension, input not a library, and any subdirectory. If the asterisk is used to specify a library check, the default file name and extension becomes UTILITY/SYS.

Directory entries are displayed in the form:

NAME/EXTENSION (PFN) P

where PFN is the physical file number in octal (0-0377) and P is the protection on the file; D for deletion, W for write, and blank for none. If the file displayed is in a subdirectory other than system, the directory entry is displayed in the form

NAME/EXTENSION-(PFN)P

with the dash indicating a subdirectory entry. All drives are searched, unless a specific drive is requested, and as each drive is scanned, the line

---- DRIVE n VOLUME ID (volid) SUBDIRECTORY (subdirectory name):

is displayed. This line is not displayed if the drive is not on line, or if no files from it are to be displayed.

Depressing the DISPLAY key causes the catalog display to pause as long as the key is held. Depressing the KEYBOARD key causes the catalog display to terminate. If CAT is already displaying it will abort immediately. If CAT is still searching it will abort when it changes drives.

If the CAT command is parameterized by an extension, only files of that extension will be displayed. If the CAT command is parameterized by a name, only files of that name will be displayed. If the CAT command is parameterized by a name and an extension, only files of that root name and extension (all drives) will be displayed. If the CAT command is parameterized by a drive number, only files on that drive will be displayed. If only a portion of the filename is entered, all files beginning with the letters specified will be displayed. For example, entering:

```
CAT /T
```

would cause the display of all files on all on-line drives whose extensions start with "T". Entering:

```
CAT MA:WORK2
```

would cause the display of all files on symbolic drive "WORK2" whose file names start with "MA".

CHAPTER 16. CHAIN COMMAND

16.1 Purpose

The CHAIN command executes a series of programs as defined by a procedure file created by the user. The procedure file contains the commands to invoke all required programs, and all inputs for those programs. Basically, CHAIN replaces the DOS keyboard entry routine with a routine that reads lines from a work file when the keyboard entry routine is called. Each time any program would normally request a line to be entered from the keyboard, it will read from the work file instead. When the last line of the work file has been read, DOS is reloaded and commands are again accepted from the keyboard.

CHAIN features several directives to control the procedure executed. Tags defined on the CHAIN command line can be specified to modify lines of the procedure file. CHAIN provides procedure restart capabilities via "CHAIN *" and "CHAIN/OV1". When used with AUTO and AUTOKEY, CHAIN provides an extensive automatic procedure facility, as described in the AUTOKEY chapter.

The procedure file is a normal DOS text format file. Procedure files are generally created using the DOS editor or the BUILD command, but may also be created by any means producing a suitable text file (a DATABUS program, for example).

16.2 Use

The command line to invoke a CHAIN procedure is of the form:

CHAIN <procedure>[;<tag_1>[=<val_1>][,<tag_2>[=<val_2>]. . .][-]]

<procedure> is the user-defined chain procedure file. This file must already exist and must be specified on the command line. The default extension is /TXT. The <tag_n> and <val_n> entries in the option field are chain tags and their substitution values, described fully below. The substitution value for a tag may be specified in the form <tag_n>#<val_n># as well as in the form <tag_n>=<val_n>.

The CHAIN command line can be extended to more than one line

by placing a hyphen (-) at the end of the option field. After scanning the current line of the command, CHAIN will display a colon as a prompt for the operator and wait for entry of another line of tags and substitution values. The command can be continued for several lines by repeated use of the hyphen.

16.2.1 CHAIN Compilation

CHAIN executes two phases, the first of which is compilation. During compilation the specified procedure file is read and compiled into a chain work file. Compilation consists of evaluating and executing CHAIN directives and performing tag substitution. The output of compilation is placed in a file called CHAINP/SYS, which directs the operation of the program chain during execution phase.

The chain work file is always placed on the same logical drive as CHAIN/CMD and CHAIN/OV1, the CHAIN program files. When operating under PS (Datapoint Partition Supervisor) the partition ID is used in the work file name instead of "P" to assure unique identification of the chain work file for each partition. The work file is placed in subdirectory SYSTEM no matter what the current subdirectory is, so the current subdirectory can be changed during the chain and the work file will still be accessible. If the work file is created on an ARC (Attached Resource Computer) remote volume it is placed in the current subdirectory (rather than SYSTEM) to avoid work file usage conflicts among different applications processors.

When CHAIN is used recursively (that is, when CHAIN is invoked from within a chain procedure) the same work file is re-used, the additional compiled information being added to the end of the file. The extent of recursive nesting of chain procedures is limited only by the amount of space available for the work file.

16.2.2 CHAIN Execution

Execution begins following compilation, when the first line of the chain work file is read and given to DOS as a command line input. Execution continues until the work file is exhausted or a fatal error occurs. During CHAIN execution the DOS keyboard entry routine is replaced by a disk read routine so that any entry normally read from the keyboard will be read instead from the chain work file. For details on this execution interface see the section on "CHAIN Programming Considerations".

CHAIN execution is aborted when:

1. A line from the chain work file is longer than allowed. DOS command lines within the chain procedure can be 80 characters long. The allowable length of lines for input to different programs depends on the programs used. For example, when a program requests a file name it generally allows about 20 characters to be entered. If a chain procedure gave a line of 30 characters in response to such a request the chain would abort.

2. The end of the work file is reached while a program is requesting input. The work file must provide all responses needed for execution of the programs used; it cannot invoke a program then end without supplying all required inputs.

3. An //ABTIF directive is executed when the ABTIF bit is set. See the section on "ABORT Directives".

4. A program executing during the chain procedure terminates in a fatal error. Each program can control whether it aborts or continues a chain upon termination. For details see the section on "CHAIN Programming Considerations".

16.3 Tag Definition

The CHAIN command line can contain both tag names and substitution values for the tags. The tag names can be from one to eight characters in length and may have values from one to seventy characters in length. A tag must contain only letters or digits. The value of a tag may contain any valid character except comma (,), equals (=) or pound sign (#). The character restriction depends on the syntax being used.

A tag is defined by just its presence on the CHAIN command line. Tags may have a value given to them by one of the following syntaxes:

```
CHAIN DOIT;LIST,DATE=30NOV76,TIME=1500hr      (New Syntax)

CHAIN DOIT;LIST,DATE#30NOV76#,TIME#1500hr#    (Old Syntax)
```

Both syntax structures are supported and the result of the two CHAIN commands is identical. The tag LIST has been defined but has a null value; DATE has the value of 30NOV76 and TIME has the value of 1500hr.

CHAIN allows two uses to be made of tags:

1. A tag can be tested to determine whether it was defined on the CHAIN command line.

2. The value of the tag can be substituted on CHAIN input statements before the line is written to the work file.

16.4 CHAIN Directives

All CHAIN directives are denoted by the characters "//" at the beginning of a line. Any number of spaces (including zero) are scanned until the CHAIN directive is reached. The first thing after the "//" must be a valid CHAIN directive else an error message is issued and CHAIN is aborted. The following is a list of these statements.

```
//IFS           IF SET (TAG DEFINED)
//IFC           IF CLEAR (TAG NOT DEFINED)
//XIF           END OF IF
//ELSE          REVERSE EFFECT OF IF
//BEGIN         BRACKETS A GROUP OF
//END             IF/ELSE/XIF STATEMENTS
//.             EXECUTION TIME COMMENT
//*             EXECUTION TIME BREAKPOINT
//ABORT         ABORT CHAIN COMPILATION
//ABTIF         CONDITIONALLY ABORT CHAIN EXECUTION
.               COMPILATION TIME COMMENT.   (Note that the //'s
                  are not present)
```

16.4.1 IF Directive

The IF directive has two variations, IFS and IFC, which are IF SET and IF CLEAR. The IFS directive proves positive if the tag named appeared on the CHAIN command line, and negative if the tag was omitted.

For example:

```
//IFS LIST
```

will prove positive if LIST was mentioned in the CHAIN command line, and negative if the tag does not exist, and

```
//IFC LIST
```

will prove positive if LIST was omitted and negative if it appeared on the CHAIN command line.

When an IF directive tests negative, it causes the chain compilation to skip all following lines of the procedure file until a directive is reached which clears the effect of the IF (an ELSE or XIF). When an IF directive tests positive it has no effect on the chain compilation. Normally the chain compilation is said to "include" lines from the procedure file; inclusion is inhibited by a negative evaluation of an IF directive.

Simple logical operations can be performed by IF directives. The tags to be used are separated by logical operators. The logical OR is indicated by '|' (vertical bar) or ',' (comma). The logical AND is indicated by '&' (ampersand) or '.' (period). For example the following lines are in the file DOIT:

```
//IFS DATE&TIME|QUICK     or     //IFS DATE.TIME,QUICK
DBCMP TEST;L                     DBCMP TEST;L
SAMPLE COMPILE                   SAMPLE COMPILE
```

If DATE and TIME or QUICK are defined on the CHAIN command line the DBCMP lines will be included in the work file.

```
     CHAIN DOIT;DATE=30NOV76,TIME=1500hr
or
     CHAIN DOIT;QUICK
or
     CHAIN DOIT;DATE,TIME
```

will all result in a true logical condition and the DBCMP lines will be included.

IF directives are only evaluated if lines are being included. If one IF directive has proven negative and has inhibited the inclusion of lines, all following IF directives will be ignored until either an ELSE or XIF statement is found. For example:

```
//IFS DATE
//IFS TIME
DBCMP TEST;L
SAMPLE COMPILATION
//XIF
```

If DATE was not defined, all lines until the //XIF will be ignored. In this example, if DATE were not defined the //IFS TIME statement would not be evaluated and the DBCMP TEST;L would not be included even if TIME was defined.

16.4.2 ELSE/XIF Directives

CHAIN has two directives that will alter the inclusion of lines from an IF directive. The first is the XIF directive. It will unconditionally terminate the range of the last IF directive. The second is the ELSE directive; it will reverse the results of the last IF directive; that is to say, if lines were being skipped because the last IF proved negative, an ELSE would cause lines to be included.

For example, the DOIT file contains the following lines:

```
//IFS LIST
DBCMP TEST;L
SAMPLE COMPILATION
//ELSE
DBCMP TEST
//XIF
//IFS TAPE
MOUT;D,30NOV76,V
TEST/DBC
*
//XIF
```

If CHAIN is invoked by 'CHAIN DOIT;LIST' the work file will contain

```
DBCMP TEST;L
SAMPLE COMPILATION
```

If invoked by 'CHAIN DOIT;TAPE', the work file will contain

```
DBCMP TEST
MOUT;D,30NOV76,V
TEST/DBC
*
```

16.5 Tag Value Substitution

A tag value is substituted whenever a pair of '#' symbols are found with a syntactically valid tag name between them. The value substituted is the tag value given in the CHAIN command line.

For example, contents of a file called DOIT:

```
DBCMP TEST;XL
TEST PROGRAM COMPILED ON #DATE# -- #TIME#
DBCMP #NAME#;XL
#NAME# PROGRAM COMPILED ON #DATE# -- #TIME#
```

If CHAIN is invoked by

```
CHAIN DOIT;TIME=2400hr,DATE=29NOV76,NAME=TEST2
```

the work file will contain

```
DBCMP TEST;XL
TEST PROGRAM COMPILED ON 29NOV76 -- 2400hr
DBCMP TEST2;XL
TEST2 PROGRAM COMPILED ON 29NOV76 -- 2400hr
```

If a tag is mentioned in the CHAIN command line but given no value and if the value is to be used for substitution, a null value is substituted for the #tag# within the line. The effect is that the #tag# characters disappear from the line. Continuing the above example, if CHAIN was invoked by

```
CHAIN DOIT;DATE=29NOV76,NAME=TEST2
```

the work file will contain

```
DBCMP TEST;XL
TEST PROGRAM COMPILED ON 29NOV76 --
DBCMP TEST2;XL
TEST2 PROGRAM COMPILED ON 29NOV76 --
```

16.6 BEGIN/END Directives

The BEGIN and END statements allow groups of IF/ELSE/XIF statements to be parenthesized. A counter called the BEGIN/END counter is initialized to zero when compilation of a procedure begins. If the use of procedural lines is turned off and a BEGIN operator is encountered, then the BEGIN/END counter is incremented. If an END operator is encountered, then the BEGIN/END counter is decremented unless it is already zero. The ELSE and XIF operators have no effect if the BEGIN/END counter is not equal to zero. For example:

```
//IFS FLAG1
DBCMP TEST1;XL
TEST PROGRAM ONE
//ELSE
//BEGIN
//IFS FLAG2
DBCMP TEST2;XL
TEST PROGRAM TWO
//ELSE
DBCMP TESTTEST;XL
TEST TESTER
//XIF
//END
//XIF
//IFS FLAG3.FLAG27
LIST SCRATCH;L
THE SCRATCH FILE AT FLAG 27
//XIF
```

The 6th through the 12th lines will not be used if FLAG1 exists, notwithstanding the fact that there is an ELSE and XIF operator within those lines, because the BEGIN/END pair prevented these statements from having any effect.

16.7 ABORT Directives

The //ABORT statement will cause CHAIN to return to DOS if it is processed. For example:

```
//IFC TIME|DATE
.**** TIME AND DATE ARE BOTH REQUIRED
//ABORT
//XIF
.
.
.
```

If the procedure file is invoked with TIME or DATE missing, the error message comment line would be displayed, and the compilation of the input file would ABORT.

The //ABTIF statement will conditionally cause the execution phase of CHAIN to ABORT. This statement causes DOSFLAG to be examined and if bit 7 (ABTIF) is on, the chaining will abort. Bit 7 of DOSFLAG is the abnormal program completion bit. If non-fatal errors have been found during the execution of the last program

the ABTIF bit should be set. For example, the procedure file contains:

```
ABTONOFF OFF
KILL TESTFILE/CMD
Y
//ABTIF
KILL OUTPUT/TXT
Y
.
.
```

If the file TESTFILE/CMD is not found by KILL, it will set the ABTIF bit. When the //ABTIF statement is processed the abnormal program completion bit will be checked, and in this case it will be on, so the CHAIN will be aborted.

The ABTONOFF command should always be used to turn the ABTIF bit off prior to execution of a program which will be tested using //ABTIF. Once ABTIF is set on by some error, it is not cleared except by ABTONOFF or by an abort caused by an //ABTIF directive.

16.8 Comments

CHAIN allows for two types of comment lines within the procedural file. One type is the execution time comment. This type may appear only before a DOS command entry and will not appear until just before that command is to be executed. An execution time comment can appear only just before a command because at any other place in a procedure file, the comment would be presented as keyboard response to an executing program. Comments can be placed at the end of a procedure, since this location is equivalent to immediately prior to a command. For example, the procedure file containing:

```
//. COMPILATION OF THE TEST PROGRAM
DBCMP TEST;XL
TEST PROGRAM
```

would cause the first line to be displayed before the assembly was executed. A variation on the execution time comment is the operator break point. For example, the procedure file containing:

```
//* INSERT TAPE Z12548 INTO THE FRONT CASSETTE DECK
MOUT ;LV
TEST/TXT
DATA/TXT
*
```

would cause a BEEP and the first line to be displayed. At this point the machine would wait for the operator to depress either the KEYBOARD or DISPLAY key and then continue with the MOUT process.

The second type of comment line is a compilation time comment. This line is not included in the work file but is displayed on the screen immediately after it is read from the procedural file. This is useful in communicating to the operator what procedure is about to be followed by CHAIN.

Both types of comment lines will be ignored (not displayed or written) just as other procedure lines if a test has proven negative and an ELSE or XIF operator has not been reached. For example, if the following procedure file MAKETEST was created:

```
. COMPILATION OF TEST PROGRAM
//IFS LIST
. YOU ARE GOING TO GET A LISTING
DBCMP TEST;XL
TEST PROGRAM
//ELSE
. YOU AREN'T GOING TO GET A LISTING
DBCMP TEST
```

and the CHAIN command:

```
CHAIN MAKETEST;LIST
```

was given, then only the lines:

```
. COMPILATION OF TEST PROGRAM
. YOU ARE GOING TO GET A LISTING
```

will appear on the screen before the procedure is executed. If, however, the CHAIN command:

```
CHAIN MAKETEST
```

was given, then only the lines:

. COMPILATION OF TEST PROGRAM
. YOU AREN'T GOING TO GET A LISTING

will appear on the screen before the procedure is executed.

16.9 Complex CHAIN Examples

The chapter on the AUTOKEY command contains an example of the use of AUTO and AUTOKEY combined with the use of CHAIN directives using tag existence testing to set checkpoints for automatic restart of a lengthy automated procedure. The example below uses BUILD within a chain procedure to create a procedure file for later execution by another chain. It uses several tags for both existence testing and value substitution.

The procedure file below, "RUNTEST", is part of a series of CHAIN procedures for program generation and testing. RUNTEST builds a procedure file for program compilation; the resulting procedure file would be run by a later CHAIN.

RUNTEST recognizes several tags:

PLUS - mention of this tag indicates the compilation should use the DBCMPLUS compiler instead of the older DBCMP compiler.

XTR - mention of this tag causes use of the additional list output commands (C and R) available in DBCMPLUS.

FLAG - the substitution value for this tag will be tag existence tested for list control on the output procedure file.

PROG - the substitution value for this tag will be a tag to provide program name in the output procedure file.

DATE - the substitution value for this tag will provide the compilation date in the output procedure file.

RUNTEST contents:

```
.
. TEST FOR DBCMPLUS COMPILER FLAG
.
//IFC PLUS
//BEGIN
.
. BEGIN PROCEDURE FOR DBCMP COMPILATION
.
BUILD COMPIT;!
.
. NOTE HOW BEGINNING INPUT LINE TO BUILD/CMD WITH THE TERMINATION CHARACTER
. ALLOWS ENTERING CHAIN COMMANDS TO THE OUTPUT FILE.  THE LINE IMMEDIATELY
. BELOW IS WRITTEN OUT AS "//IFS #FLAG#"; IF IT HAD NOT BEGUN WITH "!", IT
. WOULD HAVE BEEN INTERPRETED AS A CHAIN DIRECTIVE FOR THE CURRENT CHAIN.
.
!//IFS #FLAG#
!//* COMPILATION LISTING  -  BE SURE PRINTER IS READY
DBCMP ##PROG##;LX
##PROG## COMPILATION     #DATE#
!//ELSE
DBCMP ##PROG##
!//XIF
!
//END
.
. THIS "//ELSE" INSTRUCTION REVERSES THE EFFECT OF THE "//IFC PLUS" ABOVE
.
//ELSE
//BEGIN
.
. BEGIN PROCEDURE FOR DBCMPLUS COMPILATION USING OPTIONS OF DBCMPLUS
. BASED ON "XTR" FLAG.
. THE "BEGIN" ABOVE CAUSES THE "XIF"S AND "ELSE"S IN THE FOLLOWING SECTION
. TO AFFECT ONLY DIRECTIVES AT THE SAME BEGIN/END LEVEL, AND NOT THE
. "//ELSE" DIRECTIVE ABOVE, WHICH CONTROLS THE ENTIRE "PLUS" CONDITIONAL
. SECTION.
.
BUILD CMPLIT;!
!//IFS #FLAG#
!//* COMPILATION LISTING  -  BE SURE PRINTER IS READY
.
. THE FOLLOWING DIRECTIVES ARE RECOGNIZED DURING CHAIN COMPILATION AND
. CONTROL SELECTION OF LINES TO FOLLOW THE BUILD COMMAND ABOVE.
.
//IFS XTR
```

```
DBCMPLUS ##PROG##;LXCR
//ELSE
DBCMPLUS ##PROG##;LX
//XIF
##PROG## COMPILATION #DATE#
!//ELSE
DBCMPLUS ##PROG##
!//XIF
!
.
. PROCEDURE IS EFFECTIVELY FINISHED AT THIS POINT, BUT IT IS ESSENTIAL TO
. PROVIDE AN "END" DIRECTIVE TO MATCH THE UNMATCHED "BEGIN" ABOVE, AND
. AN "XIF" TO TERMINATE THE "ELSE" IMMEDIATELY PRIOR TO THE "BEGIN".
.
//END
//XIF . . END OF RUNTEST SAMPLE FILE .
.....................................................................
.
```

Entering the command

```
CHAIN RUNTEST;PLUS,XTR,FLAG=LIST,PROG=NAME,DATE=21OCT78
```

produces a procedure file CMPLIT/TXT with the following contents:

```
//IFS LIST
//* COMPILATION LISTING  -  BE SURE PRINTER IS READY
DBCMPLUS #NAME#;LXCR
#NAME# COMPILATION 21OCT78
//ELSE
DBCMPLUS #NAME#
//XIF
```

Entering the command

```
CHAIN RUNTEST;FLAG=PRINT,PROG=PROG,DATE
```

produces a procedure file COMPIT/TXT with the following contents:

```
//IFS PRINT
//* COMPILATION LISTING  -  BE SURE PRINTER IS READY
DBCMP #PROG#;LX
#PROG# COMPILATION
//ELSE
DBCMP #PROG#
//XIF
```

16.10 Resuming An Aborted CHAIN

Before the CHAIN overlay fetches the next DOS command it stores in the CHAINP/SYS file pointers for the line to be used. If something goes wrong during the DOS command which follows and the procedure is aborted, CHAIN still knows where it was in the CHAINP/SYS file when the problem occurred. Since CHAIN does not delete the CHAINP/SYS file unless the procedure completes successfully, it can pick up where it stopped in the CHAINP/SYS file if the operator can correct the condition which caused the procedure to abort in the first place. Often, the reason for the abort is something correctable like the disk running out of files. In this case, the operator need only correct the condition and then enter:

```
CHAIN *
```

and the procedure will pick up with the command which failed before. This action can generally be applied even if the RESTART key has been depressed. Thus, one can recover from jammed paper in a printer half way through a listing by simply depressing RESTART, fixing the printer, and then entering the CHAIN * command.

If the failing command cannot ever succeed, it may be bypassed by entering the command:

```
CHAIN/OV1
```

This simply restarts the chain with the next available line in the procedure. If the next line had been intended as a keyin line for the failed program (as opposed to a DOS command line) the chain will generally immediately abort again. However, by restarting the chain in this manner, repeatedly if necessary, the invalid step can usually be bypassed and chaining resumed. Use of CHAIN/OV1 will not always work, since it depends on information in processor memory to function. If the area from MCR$+80 to MCR$+100 is disturbed, CHAIN/OV1 will fail, usually causing a range error or perhaps a system data failure.

16.11 CHAIN Programming Considerations

CHAIN only replaces the DOS keyboard entry routine (KEYIN$). Therefore, only programs that use this routine for input will receive their input from the chain file. Programs which have their own input routines, like the DOS editor, can be invoked from a chain file but editing must be done manually by the operator. Sometimes programs will use a different keyin routine based on DOS Function 6 to request operator action for special circumstances when it is desired to avoid using lines from the chain procedure.

When a program exits via EXIT$ or NXTCMD the chain continues normally. If a program exits via ERROR$ or CMDAGN the chain is aborted. Generally the terminating error message displayed by an aborting program will remain visible on the screen following the CHAIN abort.

Some programs can go through a rather complex set of requests for input, which can make them difficult to use with the CHAIN program. For this reason, most DOS programs allow almost all options to be specified on the command line and keep the variation in the number of keyin requests to a minimum. It is good practice for all programs to be written with this concern in mind to facilitate their use with CHAIN.

CHAPTER 17. CHANGE COMMAND

17.1 Purpose

The CHANGE command enables one to write protect, delete protect, or clear the protection of a disk file. If a file is delete or write protected, a KILL command (or program generated KILL) cannot affect it. If a file is write protected, it cannot be written into by the standard system routines.

17.2 Use

CHANGE <file spec>;p

The option parameter "p" is used above to indicate the protection for the file specified. Protection can be specified as:

D - delete protect
W - write protect
X - clear protection.

For example:

```
CHANGE NAME/EXTENSION;D
CHANGE NAME/EXTENSION:DR2;X
```

will delete protect the file in the first case, and remove all protection in the second case. If a <file spec> is not given, the message

NAME REQUIRED.

will be displayed. If the file indicated by the first file specification cannot be found, the message

NO SUCH NAME.

will be displayed. If the option parameter does not follow the above syntax rules, the message

INVALID PROTECTION SPECIFICATION.

will be displayed. If no option parameter is specified the message

PROTECTION UNCHANGED.

will be displayed.

CHAPTER 18. COPY COMMAND

18.1 Purpose

The COPY command produces a duplicate copy of a disk file. It may be desired, for example, to make a copy on a separate volume for backup or distribution purposes.

Another feature of the COPY command will optionally allow a user to selectively update (replace) an existing file, or create (add) a new file to receive the copy. These options used in combination with the CHAIN utility provide an easy method of updating and maintaining DOS disks.

The COPY command does not make assumptions about the format of the sectors being copied, but merely copies the file sector-for-sector. It can copy types of disk files which are not possible to copy using the SAPP and APP commands. Some particular types of files are still immovable, however. The outstanding example are INDEX files, usually with extension /ISI. These files cannot be moved because index files contain, internal to themselves, pointers indicating their actual physical location on the disk volume, which are made invalid when the file is moved to another place on the disk.

18.2 Use

The COPY command is invoked by entering at the system console:

COPY <infile>[,<outfile>][;<option>]

The only portion of the operands that is specifically required is the name of the input file. The extension of the input file, if none is specified, is assumed to be /TXT. If a drive specification is entered for the input file, then only that specific drive is searched for the indicated file. If no drive specification for the input file is given, all drives are searched. If the name of the output file is omitted, it is assumed to be the same as that of the input file. If the output file's extension is not given, it is also assumed to be the same as that of the input file. All drives are searched for the output

file unless a particular drive is specified.

The options available are:

R - Replace only
This option allows only overwriting an already existing file. If the specified output file does not already exist no copy takes place, the message "<infile> NOT COPIED" is displayed and the ABTIF bit is set.

A - Add only
This option allows only creating a new file. If the specified output file already exists no copy takes place, the message "<infile> NOT COPIED" is displayed and the ABTIF bit is set.

E - Chop on end of file
The output file will be chopped at the first EOF mark found in the input file.

7 - SYSTEM7/SYS to be copied
This option allows overwriting an existing copy of the system file SYSTEM7/SYS with a different copy, without disturbing the subdirectory information stored in the old SYSTEM7/SYS. This option is very handy for system upgrades. The output file must be PFN 7.

If no options are specified there are no special conditions on the copy. The input file will be copied to the output file, overwriting an existing file if necessary (unless the output file is write protected). If the specified output file does not exist, it will be created.

When a file is transferred via COPY, the output file is set to the same protection that the input file had.

Example, to copy file PAYROLL/TXT from symbolic drive "WORK2" to symbolic drive "WORK1"

```
COPY PAYROLL:WORK2,:WORK1
```

Example, to make another copy of PROGRAM/ABS on drive zero, but to be named MYPROG.

```
COPY PROGRAM/ABS,MYPROG:DR0
```

Example, to make another copy of PAYROLL/TXT drive 0, on drive 1 only if it does not already exist on drive 1.

```
COPY PAYROLL:DR0,:DR1;A
```

Example, to update (replace only) TREK/ABS, a file on drive 0 from a newer version on drive 1.

```
COPY TREK/ABS:DR1,:DR0;R
```

People who experience parity errors in one of their data files can frequently recover their data using COPY. Since the COPY program merely comments about parity errors encountered and does not abort when one occurs, the data copied will occasionally be correct (or almost correct) even if a parity error occurs and can be used to recover the data in the original file. Alternatively, using the COPY program to write the file on top of itself (therefore without changing the file) by simply specifying the input file and no output file, a user can frequently clear soft (and occasionally what seem to be hard) parity errors occurring in an important data file. (Of course, no important file should be updated in place unless a copy of the file exists somewhere for recovery purposes in the event of a failure.)

The COPY command issues a click each time an unused sector is copied. If more than a dozen or so clicks occur at the end of copying a file, it usually indicates that the file is larger than necessary to contain the data in it. In this case, moving the file using APP or SAPP can sometimes help to reduce its size. If the KEYBOARD key is depressed while the machine is clicking, the output file will be chopped at that point. Clicks ocurring during the copying (before the end of the file) indicate sectors containing DOS format errors, possibly implying a sector accidentally destroyed by some faulty program.

CHAPTER 19. DOSGEN COMMAND

19.1 Purpose

Before any disk can be used by DOS, certain tables and other information must be placed onto it to establish the basis that DOS requires for the support of its file structure. These tables include the skeleton of the DOS directory, (where the names of the files contained on the disk are stored), as well as a map showing which places on the disk are bad and should not be used.

The purpose of the DOSGEN command is to provide the user with a simple way of accomplishing this preparation.

19.2 Use

To DOSGEN a disk enter:

DOSGEN <drive spec>

The drive spec is a standard DOS drive specification which specifies which drive contains the disk to be prepared for DOS use. Since the directory initialization process will effectively KILL any files that might be on the disk, the command asks several times to make sure that the operator is aware of the potential seriousness of the operation he has invoked.

After the operator has acknowledged that he does not mind the overwriting of the new disk, the command asks if any cylinders on the volume are to be locked out. Normally, the answer to this question is NO. However, by answering YES, it is possible to cause the DOS to lock out one or more cylinders of the disk from DOS access. This can be useful in some special applications where it is desired to not allow DOS programs access to a file stored in unusual format. If the user does wish to lock out any cylinders, he may do so by specifying one or more cylinder numbers, in the format:

12,14,16,25-28,40

The above example would cause cylinders 12, 14, 16, 25, 26, 27, 28, and 40 to be locked out. The cylinder numbers to be

locked out are entered in decimal.

After the operator has specified that no, or which, cylinders are to be locked out, the DOSGEN command checks for bad sectors on the disk and issues a message indicating any cylinders it finds which contain bad sectors. Any cylinders found bad are automatically locked out and will not be used by DOS. The remainder of the operation is completely automatic and indicates its completion with the DOS "READY" message.

Upon completion of the DOS generation process, the only files on the new disk are the eight system files SYSTEM0/SYS through SYSTEM7/SYS. In the case of DOS.D and DOS.E, UTILITY/SYS, UTILITY/REL and UTILITY/LNK are also copied.

19.3 Special Considerations

It is important to remember that on disk packs for use with DOS systems recognizing more than one logical drive per physical disk pack, for example the 9370 and 9390 series disk systems, more then one DOSGEN must be done before the physical pack is fully initialized. This allows the user to DOSGEN any logical disk on the pack without disturbing files he wishes to keep that may be stored on other logical disks.

Another important thing to remember is that the 9370, 9380 and 9390 series disks must be formatted before DOSGEN can be used on them. Diskettes (for the 9380 series drives) come pre-formatted from the manufacturer. A diskette that has been formatted with tracks locked out (error mapped) cannot be DOSGENed. Disk packs for the 9370 and 9390 series drives do not have formatting when purchased. It is therefore necessary to format all disk packs for the 9370 or 9390 series drives using the program INITDISK before attempting to use DOSGEN on them.

The booted drive of a system cannot be DOSGENed. PUTIPL must reside on the booted drive. Following surface write/read verification, the system files are copied. COPY and CHAIN are no longer required.

CHAPTER 20. DSKCHECK

20.1 Purpose

The purpose of DSKCHECK is to repair a logically-damaged DOS (Disk Operating System) volume. The performance of the DOS is directly related to the correctness of disk-resident system tables. DSKCHECK checks all system tables for format and content and is able to determine in most cases when an error in the system tables occurs, what the error is, and if the system tables can be reconstructed from the other data on the disk.

20.2 Use

DSKCHECK is invoked by entering:

DSKCHECK [<drivespec>][;<options>]

The drive specification may be entered as a standard drive specification such as :D0, or as a VOLID (volume identification) such as :PAYROLL. If no drive specification is entered on the command line, the program will ask for one.

20.3 Options

Option letters that may be used are "L", "S", and "F". Each is explained in the following paragraphs. To be activated, an option must be entered on the command line.

L Option, Local printer on: With this option keyed in, all messages will be printed on the local printer as well as displayed on the screen. An option conflict message will appear if both the "L" option and "S" option are selected.

S Option, Servo printer on: With this option, all messages will be printed on the Servo printer as well as displayed on the screen. If "S" and "L" are both selected, an option conflict message will appear.

F Option, System Fix option: This option causes the program

to compute the correct data when an error in system data is detected, if such a computation is possible, and allows the user to fix the data which is in error if he so chooses.

If an error in system data is detected and the correct data cannot be computed from the other pertinent usable data on the disk, the operator is so informed and may be asked if he wishes the file or entry deleted.

All error-correction messages contain a no-change option which, if elected, cause the program to continue to the next check without changing the data on the disk being checked. This no-change option is illegal and automatically deactivated if DSKCHECK is run from CHAIN, under ARC, or under PS.

If no options are given on the command line, the default condition is logging to the screen only. The Fix option is turned off.

20.4 System Tables and Data

Descriptions of the DOS tables, system data, and their uses are found in the chapter on System Structure.

All of the descriptions below are written as if the "F" (fix) option were set. If this option is not active, the operation is the same except that no request for corrective action is made, if this is applicable.

20.5 Execution Phases

There are many execution phases in DSKCHECK, some of which are dependent on the type of DOS in use or on the existence of previous conditions. Some of the executuion phases may or may not be performed.

20.5.1 Initialization

During initialization, the program displays a signon message and the options parameters are scanned. If the "F" (fix) option was selected, the correction features are enabled. If the <drivespec> is not specified on the command line, it is asked for during this phase.

20.5.2 HDI Checking

This phase of checking is dependent on the type of disk operating system residing on the disk being checked. It is not executed on DOS.C because this DOS does not use the HDI (Hashed Directory Index) technique of directory indexing. DOS.C uses directory mapping, which is verified in a later phase.

This phase reads the HDI on the disk and compares the master to the backup. If an error occurs in reading either copy, or in comparing the copies, an error message will be displayed, but no corrective action is allowed at this time. If no errors are detected, an appropriate message is displayed.

20.5.3 CAT Checking

This phase of checking reads the CAT (Cluster Allocation Tables) on the disk being checked. The master CAT is compared with the backup, and the same procedure described above for HDI checking is followed. Appropriate messages are displayed, but no corrective action is permitted during the checking phase.

20.5.4 Directory Checking

During this phase each directory page is read and checked. The master is compared to the backup copy, and if the pages do not match, the affected entry is displayed. The program then asks which entry should be retained, or if both entries should be deleted.

When an unsuccessful read occurs in either the master or backup, an error message is displayed and the comparison is not done. The good page is used to continue the checks. If both pages fail to read successfully, an appropriate message is displayed and the next page is checked.

If either or both pages read successfully, and when all compare errors are resolved, each entry is checked for valid format. If a deleted entry is encountered, it is checked to see that the delete is complete. If it is not, an error message is displayed and the program asks if the error is to be corrected.

The RIB PDA (Retrieval Index Block, Physical Disk Address) is checked to see that it points to a valid location on the disk. If an error is detected, an appropriate message is displayed, and the program asks if the entry should be deleted. If this entry is deleted, no space for it will be allocated in the cluster allocation tables when the CAT

is re-computed.

The program then proceeds by checking the filename/extension for valid characters. Each character should fall into the range of A-Z or 0-9. If the program finds an invalid character, a warning message is displayed and the program continues.

Under DOS.C, the program generates a mapping byte for each page as the page is checked. If the computed mapping byte does not match the mapping byte on the disk, a message is displayed and the newly-computed mapping byte is entered into the appropriate sector of the CAT that is being generated.

20.5.5 RIB Checking

After all directory pages have been checked, the RIB for each entry in the directory is checked for format and space allocation validity. The RIB master and backup are first read and compared. If either copy is not read successfully, the program asks if the good copy is to be written to the copy with the read failure. If a read failure occurs on both copies, an error message will inform the user and the program will ask if the file is to be deleted. No space will be allocated for that file in the CAT that is being re-computed.

If a compare error is detected, the filename and the first 16 bytes of both copies are displayed; the program will ask which copy is to be retained on the disk.

After the RIB master and backup have been read and compared and all errors resolved, the RIB format is checked for the correct PFN (Physical File Number), LRN (Logical Record Number), and an 0377 (indicating the end of the retrieval information block in the directory sector) in the fourth byte. Any format error detected will be displayed, and the corrective action reuested.

Each segment descriptor is then checked to see that it points to a valid location on the disk, and that the first sector of the segment has the proper PFN. The space allocation is computed and checked against the CAT and the Lockout CAT (discussed later) for conflicts. If a conflict occurs with the Lockout CAT, the program will ask if the Lockout CAT is to be rewritten to free this space. If a conflict is detected with another file, the conflicting area is analyzed in the next two phases and is displayed with a request for action. If no conflicting areas are found, the next two phases are skipped and the program continues with the Lockout CAT check.

20.5.6 Cluster Allocation, Phase 1

If a conflict was detected while checking the RIBs, this phase is activated to re-read all the good RIBs and find all other conflicts.

20.5.7 Cluster Allocation, Phase 2

Upon the determination of all conflicting clusters, they are scanned to see which RIBs conflict. The program gives an opportunity to delete one or the other or both conflicting files. If no change is requested at this point, the CAT is updated with the conflicting clusters to retain protection from another file writing into this area.

20.5.8 Lockout CAT Checking

If no change in the Lockout CAT has been made previously, it is now read to see that all locked out space is also allocated in the CAT. If changes were made in previous phases of the program, the generated Lockout CAT is checked against the CAt; if no errors are detected, it is written to the disk.

Any errors detected in this phase cause a descriptive message to be displayed and the program asks if a new Lockout CAT should be written. This new Lockout CAT, if written, will only have the space for the system tables locked out.

20.6 Operational Messages

The messages listed in this section are informational. They are displayed to indicate where the program is in the execution cycle.

CHECKING HDI

> The program is checking the HDI master and backup for read errors in either, and for a match between the two.

CHECKING CAT FOR FORMAT

> The program is reading the CAT master and backup and checking for read errors in either, and for a match between the two.

CAT MASTER AND BACKUP LOOK O.K.

The CAT master and backup were read without error, and the master compared correctly with the backup.

CHECKING DIRECTORY PAGE nnn

The directory page nnn master and backup have been read without error, and the directory entries on this page are being checked for format. During this check, a directory mapping image is also being constructed and checked against the one on the disk, if necessary.

CHECKING RIBS

The directory checking is complete, and the RIBs are not being checked.

CHECKING RIB FOR PFN nnn

The RIB for PFN nnn is now being read and checked for format errors and allocation conflicts.

CLUSTER ALLOCATION PHASE 1

If cluster conflicts occur while checking RIBs and the 'F' option is set, This phase is executed to find and verify all conflicts.

CLUSTER ALLOCATION PHASE 2

This phase follows the phase 1 above to resolve any or all conflicts.

CHECKING LOCK-OUT CAT

The Lockout Cat master and backup are being read and compared, and checked to be sure that the locked-out space is allocated in the CAT.

ALL LOCKED-OUT CYLINDERS ARE ALLOCATED IN THE CAT

The display indicates that no Lockout CAT errors exist and that all cylinders locked out are allocated in the CAT.

MATCHING COMPUTED CAT TO DISK

The CAT computed from the RIBs and the computed directory mapping bytes is being compared against that read from the disk.

COMPUTED CAT MATCHES DISK

The CAT computed from the RIBs matches the CAT on the disk.

COMPUTED HDI MATCHES DISK

The HDI read from the drive is complete and control is being returned to the DOS.

20.7 Error Message Definitions

Error messages for DSKCHECK follow.

20.7.1 Operational Error Messages

OPERATOR INTERVENTION - JOB TERMINATED

If the KBD (keyboard) key is depressed at any time during the operation of the progam, this message is displayed and DSKCHECK is terminated.

****WARNING**** WRITE FAILURE

If at any time a write to disk is attempted and does not complete successfully, a warning is given. This is a serious error because it means that an error found by DSKCHECK was not corrected. However, the program will allow the operator to discover if there are any other errors on the disk, and it is then the operator's responsibility to recover as much of the data as possible.

nnn WRITE FAILURES HAVE OCCURRED!!! PLEASE NOTE

If a write failure did occur, this message is displayed at the termination of the program to re-alert the operator to the fact that something must be done about it.

20.7.2 Initialization Error Messages

DRIVE OFF LINE

When accessed, the drive being checked was found to be offline.

INVALID DRIVE

The drive selected was not a valid number or not in valid format.

INVALID OPTION PARAMETER
VALID OPTIONS ARE L=LOCAL S=SERVO F=FIX

An option other than those listed was detected.

MULTIPLE PRINTERS SELECTED

Both servo and local printer were selected.

PROGRAM NOT LOADABLE

One or more of the library members of DSKCHECK are missing. The members in DSKCHECK are:

DSKCHECK
DSKCVER1
DSKCVER2
DSKCVER3
DSKCVR3B
DSKCVR3C
DSKCVER4

RELOCATABLE MEMBER MISSING OR UNLOADABLE

UTILITY/REL FILE MISSING

SECTOR IN/OUT MEMBER OF UTILITY/REL IS WRONG VERSION, I NEED VERSION x

PRINT DRIVER MEMBER OF UTILITY/REL IS WRONG VERSION, I NEED VERSION x

PRINT MODULE MISSING OR NOT LOADABLE. NO PRINT WILL OCCUR.

All the above messages show different ways of checking that UTILITY/REL members are missing or unloadable.

YOUR DOS FUNCTION 15 IS OBSOLETE

The processor is running on an obsolete DOS that does not recognize DSKCHECK.

WRONG DOS

The processor is running on an obsolete DOS that does not recognize DSKCHECK.

THIS PROGRAM IS RUNNING IN A CHAIN - THE "F" OPTION HAS BEEN DEACTIVATED

The fix option is not allowed during CHAINing.

THIS PROGRAM IS RUNNING UNDER ARC - THE "F" OPTION HAS BEEN DEACTIVATED

The fix option is not allowed while on ARC.

THIS PROGRAM IS RUNNING UNDER PS - THE "F" OPTION HAS BEEN DEACTIVATED

The fix option is not allowed while under PS.

ENTER DRIVE (LIKE :D0, :D1, OR :VOLID)

An invalid drive specification or no drive specification was detected from the command line.

When an asterisk (*) is entered, control returns to the DOS.

20.7.3 HDI Errors

HDI MASTER AND BACKUP DO NOT MATCH
HDI WILL BE RECONSTRUCTED FROM DIRECTORY

The Hashed Directory Index master and backup did not match and will be recreated during the final phase.

ERROR ON READ OF HDI MASTER

Parity error found while reading the HDI master.

ERROR ON READ OF HDI BACKUP

Parity error found while reading the HDI backup.

WRITE NEW HDI TO DISK?

With the fix option set, the operator will be able to try to write a new HDI.

20.7.4 CAT Errors

ERROR ON READ OF CAT MASTER

An unrecoverable read error occurred during the read of the master CAT sector. No action is taken at this time.

ERROR ON READ OF CAT BACKUP

An unrecoverable read error occurred during the read of the backup CAT sector. No action is taken at this time.

CAT MASTER AND BACKUP DO NOT MATCH
CAT WILL BE RECONSTRUCTED FROM THE RIBS

The CAT master and backup buffers in memory do not match. No action is taken at this time. When a computed CAT has been constructed, it will be compared to the master and backup CAT, and at that time corrective action may be taken.

Before the computed CAT is compared to the disk, the program checks internal error flags, and if any of the errors above are found to have occurred, one or more of the following messages may be displayed. Usually it is best to answer "Y" so the CAT on disk will be corrected.

ERROR ON READ OF CAT MASTER
WRITE BACKUP TO MASTER?

An error was detected on the initial read of the CAT master, but the backup copy was read successfully. Answering "Y" causes the backup to be written to the master. The "N" answer causes no change.

ERROR ON READ OF CAT BACKUP
WRITE MASTER TO BACKUP?

An error was detected on the initial read of the CAT backup, but the master copy was read successfully. Answering "Y" causes the master to be written to the backup. The "N" answer causes no change.

BOTH CAT COPIES BAD
WRITE COMPUTED CAT TO BOTH?

Neither copy of the CAT could be read successfully. Answering "Y" causes the computed CAT to be written to the CAT master an backup sectors on the disk. The "N" answer results in no change.

CAT MASTER AND BACKUP DID NOT MATCH
WRITE COMPUTED CAT TO DISK?

After the initial read, the CAT master and backup did not match. Answering "Y" causes the computed CAT to be written to the CAT master and backup sectors on the disk. The "N" answer results in no change.

NOTE: Answer "Y" to get a good disk.

20.7.5 Directory Errors

The following error messages may appear when the program is checking the directory. Some error conditions refer to the HDI, and will not appear when checking a disk without an HDI. Likewise, those messages that refer to directory mapping bytes will not appear on disks that do not use directory mapping.

ERROR ON READ OF DIRECTORY MASTER

The current directory master page could not be read successfully. If the "F" (fix) option is active, the request "WRITE BACKUP TO MASTER?" will also appear. If this request is answered "Y", the current backup page will be written to the correct directory master sector on the disk.

ERROR ON READ OF DIRECTORY BACKUP

The current directory backup page could not be read successfully. If the "F" option is active, the request "WRITE MASTER TO BACKUP?" will also appear. If this request is answered "Y", the current master page will be written to the correct directory backup sector on the disk.

BOTH PAGES READ BAD
NO SPACE WILL BE ALLOCATED FOR THE FILES ON THIS PAGE

Neither the master nor backup pages could be read successfully. The page cannot be fixed and no space will be allocated in the computed CAT for the files on this page, nor will HDI entries for the files on this page appear in the computed HDI.

DIRECTORY ENTRIES DO NOT MATCH - PFN nnn

The current master and backup page do not match. The master and backup copies of the entry that do not match will be displayed in the format below.

DIRECTORY ENTRIES DO NOT MATCH - PFN 000

```
MASTER -
  001 303 000 000 123 131 123 124 105 115 062 040 123 131 123 377
                   S   Y   S   T   E   M   2       S   Y   S

BACKUP -
  001 003 000 000 123 131 123 124 105 115 060 040 123 131 123 377
                   S   Y   S   T   E   M   0       S   Y   S
ACTION:  1=MASTER>BACKUP 2=BACKUP>MASTER 3=DELETE BOTH 4=NO CHANGE?
```

One of the following lines will also be displayed, indicating the relationship between the filenames in the directory entries and the value in the HDI:

```
NEITHER MASTER NOR BACKUP MATCH HDI; RIB WILL NOT BE CHECKED.
```

or

```
MASTER ENTRY MATCHES HDI
```

or

```
BACKUP ENTRY MATCHES HDI
```

```
DELETE INCOMPLETE PFN nnn
WRITE DELETE TO DIRECTORY?
```

The entry for PFN nnn was found to have one or more bytes other than 0377. The "Y" answer causes a deleted entry to be written to the directory for this entry. The "N" answer causes no change.

```
INVALID PDA IN PFN - nnn
DELETE THE ENTRY?
```

The entry for PFN nnn points to a physical disk address that is not valid for a RIB location. The "Y" answer causes the entry to be deleted from the directory master and backup. The "N" answer causes no change.

```
COMPUTED HASH BYTE DOES NOT MATCH HDI FOR PFN nnn
```

The computed hash character for PFN nnn does not match the hash character on the disk. No corrective action is taken, but the

computed hash character is entered into the computed HDI, and this will cause an error when the computed HDI is compared to the disk.

WARNING FILENAME/EXT FOR PFN nnn CONTAINS INVALID CHARACTERS

The filename/extension for PFN nnn contains characters that are not in the range A-Z or 0-9. This is a warning only, and does not allow any corrective action.

DIRECTORY MAPPING BYTE DOES NOT MATCH GENERATED BYTE FOR PAGE nnn

The directory mapping byte generated for the current page (nnn) does not match the mapping byte on the disk. The generated byte is entered into the directory mapping area in the computed CAT and will cause a match error when the computed CAT is compared to the CAT on the disk.

20.7.6 RIB Errors

The error message and display formats that follow may all be displayed in the RIB checking phase of program execution.

ERROR ON READ OF RIB MASTER
ACTION: 1=BACKUP>MASTER 4=NO CHANGE?

The RIB master could not be read successfully, but reading the backup was successful. The "1" answer causes the backup RIB to be copied to the master with the correct LRN (logical record number). The "4" answer causes no change on the disk, and the backup RIB is used for further checks.

ERROR READ OF RIB BACKUP
ACTION: 2=MASTER>BACKUP 4=NO CHANGE?

The RIB backup could not be read successfully, but reading the master was successful. The "2" answer causes the master RIB to be copied to the backup with the correct LRN. The "4" answer causes no change on the disk.

MASTER AND BACKUP READ ERRORS

```
DELETE THE FILE?
```

Neither copy could be read successfully. No recovery of data is possible, and the file may be deleted from the disk, or no change may be made. Either action results in no space for this file being allocated in the computed CAT.

```
RIB MASTER AND BACKUP DO NOT MATCH *** FILE - filename/ext
RIB MASTER -
  000 000 000 377 001 004 377 377 377 377 377 377 377 377 377 377
RIB BACKUP -
  000 001 000 377 001 104 377 377 377 377 377 377 377 377 377 377
ACTION:  1=MASTER>BACKUP 2=BACKUP>MASTER 3=DELETE BOTH 4=NO CHANGE?
```

The RIB master and backup copies did not compare correctly. Only the area of the RIB used by the system is compared. The responses are self-explanatory. For any action except "3", the resulting master is used for checking allocation. If the file is deleted, no further checks are made, and no space is allocated in the constructed CAT.

```
RIB FORMAT ERROR
```

One or more of the first four bytes of the RIB was incorrect. The total message will display in the format below. Only the error explanations and pointers for which errors were detected will be displayed.

```
RIB FORMAT ERROR *** FILE - filename/ext
000 123 125 105 000 377 377 377 377 377 377 377 377 377 377 377
 ^   ^   ^   ^
 1   2   3   4

1 - PFN INCORRECT
2 - LRN LSB NOT 000
3 - LRN MSB NOT 000
4 - 4th BYTE NOT 0377

ACTION: 1=DELETE THE FILE 2=CORRECT ERROR(S)  3=NO CHANGE?
```

RIB BACKUP LRN ERROR
CORRECT THE ERROR?

The RIB backup LRN was not 001, but the rest of the backup matched the RIB master. The "Y" answer causes the correct LRN to be written to the RIB backup sector.

INVALID SEGMENT DESCRIPTOR # nnn - RIB WILL NOT BE CHECKED

The segment descriptor for segment nnn pointed to a location that cannot physically exist on the disk being checked. No further checking of the RIB will occur.

SPACE ALLOCATION CONFLICTS WITH PREVIOUS ALLOCATION

The space allocated for the current segment of this file is in conflict with space allocated to a previously-checked file. Additional information is displayed in the format below.

```
                  FILENAME/EXT   *  FILENAME/EXT
                      PFN  014   *       143
CONFLICTING CLUSTERS           007   *       007
CLUSTERS IN FILE               005   *       002
```

ACTION: 1=DELETE PFN 014 2=DELETE PFN 143 3=NO CHANGE?

The above indicates that PFN 014 and PFN 143 have 7 clusters in conflict, and that 5 of them have records from PFN-014, and 2 have records from PFN-143. In this example, PFN-143 should probably be deleted.

CURRENT FILE OVERLAYS LOCKED OUT SPACE
FREE THIS SPACE IN LOCKOUT CAT?

Some of the space allocated to the file being checked occupies space that is locked out. Answering "Y" causes the cylinders in the Lockout CAT which the file overlays to be removed from the Lockout CAT. The "N" answer causes no change in the Lockout CAT.

INVALID PFN IN FIRST SECTOR OF SEGMENT nnn *** FILE - filename/ext

The PFN in the first sector of segment nnn does not match the

PFN of the file being checked. This message is informational, as this may be a normal condition in some files. The program will take no corrective action.

20.7.7 Lockout CAT Errors

LOCKOUT MASTER AND BACKUP DON'T MATCH
WRITE NEW LOCKOUT CAT TO DISK?

Tne Lockout CAT master and backup do not match. Answering "Y" causes a Lockout CAT with only the space for the system tables locked out to be written to the disk. The "N" answer causes no change.

LOCKED OUT CYLINDER nnn NOT ALLOCATED IN CAT
WRITE NEW LOCKOUT CAT TO DISK?

The locked out cylinder nnn is not allocated in the CAT. Answering "Y" causes a Lockout CAT with only the space for the system tables locked out to be written to the disk. The "N" answer causes no change.

ERROR ON READ OF LOCKOUT CAT
WRITE NEW LOCKOUT CAT TO DISK?

The Lockout CAT read was bad, possibly a parity error. The user now has the opportunity to try to write the Lockout CAT calculated back to disk.

CHAPTER 21. DUMP COMMAND

21.1 Purpose

The DUMP command provides a simplified mechanism for examining the entire contents of physical sectors on the disk. The display includes both the octal and ASCII contents of every byte in the sector. No examination for control bytes of any kind is made, allowing the user to see the precise contents of every physical location in the disk sector.

21.2 Use

The DUMP command is invoked by entering:

DUMP [<filespec>][;<options>]

The DUMP command operates with basically five separate levels of control. These levels are:

LEVEL ONE - Logical drive level
LEVEL TWO - File level
LEVEL THREE - Logical record number level
LEVEL FOUR - Physical disk address level
LEVEL FIVE - Disk directory level

The entry file and/or drive specifications on the command line allow the first one or two input levels in DUMP to be automatically bypassed.

The <option> is the type of printer the operator wishes to have his printing done on. The type printers supported are either local or servo. The option letter 'L' is used for local and 'S' for servo. The default is no printer. The 'P' option may be used to create a print file or the 'Q' option to queue to an existing print file. If either the 'P' or 'Q' options are used the system will prompt for a print file specification.

When the DUMP command is used, the top line on the display is the primary control line. Input is accepted on this line. This line is broken into four basic areas, one corresponding with each of the first four control levels. The primary control level at

any given time during the operation of the DUMP command can be determined by the position of the flashing cursor on the control line.

For example, if the flashing cursor is positioned after the "DRIVE:" legend on the control line, the DUMP command is operating at level one. If the cursor is positioned after the "FILE:" legend on the control line, the DUMP command is operating at level two, etc.

21.3 Informational Messages Provided

The second line on the display is primarily used for sector informational messages. These serve both to indicate any special significance of the sector just read and to describe any unusual occurrences associated with reading the sector. These messages are generally self-explanatory. Among the messages that can be displayed are the following, along with an explanation of the meaning of each.

RETRIEVAL INFORMATION BLOCK (RIB). This message indicates that the sector being displayed is the primary RIB for the currently opened file.

RETRIEVAL INFORMATION BLOCK BACKUP. Each RIB is maintained in duplicate for backup purposes and to allow recovery in the event of a program erroneously destroying the primary RIB. This message indicates that the sector being displayed is the secondary RIB for the currently opened file.

CLUSTER ALLOCATION TABLE. This message indicates that the sector being displayed is the primary Cluster Allocation Table (normally referred to as the CAT) for the current logical drive.

CLUSTER ALLOCATION TABLE BACKUP. This message indicates that the sector being displayed is the secondary, backup CAT for the current logical drive. The CAT is also maintained in duplicate just as is the RIB.

LOCKOUT CLUSTER ALLOCATION TABLE. Associated with each logical drive is a sector that indicates which areas have been locked out, prohibiting their use by DOS. This message indicates that the sector being displayed is the Lockout CAT for the current logical drive.

LOCKOUT CLUSTER ALLOCATION TABLE BACKUP. This message indicates that the sector being displayed is the secondary, backup

copy of the Lockout CAT.

SYSTEM DIRECTORY SECTOR. This message indicates that the sector being displayed is one of the DOS directory sectors. The directory sector number (in decimal and in octal) immediately follows the message.

USER DATA SECTOR. This message indicates that the sector is not recognized as one of the above special system sectors.

DISK SECTOR CRCC ERROR. This message indicates that the sector requested for display either was not found on the disk or that a CRCC error repeatedly occurred during the read operation. The sector displayed is the data as it was read from the disk, unless the sector was not found.

DISK OFFLINE. This message indicates that the currently specified logical drive is not on line.

DISK SECTOR FORMAT ERROR. This message is displayed when DUMP notices that the sector being displayed does not correspond to standard DOS file conventions (the first byte of each sector is its physical file number, and the two following bytes are the logical record number). The appearance of this message does not necessarily indicate that the sector of the file has been destroyed, since unwritten sectors at the end of a file and older version DATASHARE object code files normally will fall into this class. It merely means that if the sector were read with the DOS READ$ routine, a format trap would occur.

SECTOR OUT OF RANGE. This message is displayed if the sector requested (by logical record number) is not within the range of the currently opened file.

FILE NOT FOUND. This message indicates that the file requested could not be found. This does not necessarily mean that the file does not exist. For example, the file could be in a non-current subdirectory. If the user has not requested non-specific volume mode (to be described), this message might mean simply that the file desired is on a different logical drive.

INVALID PHYSICAL ADDRESS. This message indicates that the physical disk address specified is invalid.

The remainder of the display contains the contents of the current half of the sector most recently read. The display is arranged as eight groups of sixteen bytes each. Each of these groups is preceded by the three octal digit offset of that group

within the sector. Each sixteen byte group consists of the octal and ASCII contents of each of the sixteen bytes in that group. Each byte's contents form a column one character wide and four lines high, where the first three lines are the value of the byte, in octal, and the fourth line is the ASCII value of that character. Notice that the character is not examined for special significance before it is displayed, so that computers having the high speed RAM display option (which is strongly recommended for all DOS systems) may display characters other than the normal ASCII set.

21.4 Level One Commands To DUMP

When the flashing cursor indicates that DUMP is functioning at level one, the following commands are accepted:

<enter> - The CAT on the current drive is displayed and control is transferred to level two. In addition, the non-specific drive mode is enabled.

number - The drive number indicated becomes the currently selected drive. The CAT from that drive is displayed and control is transferred to level two. Non-specific drive mode is disabled.

* - DUMP command returns control to the DOS.

> - The second half of the current sector is displayed.

< - The first half of the current sector is displayed.

21.5 Level Two Commands To DUMP

When the flashing cursor indicates that the DUMP command is functioning at control level two, the following commands are accepted:

<enter> - If a file is currently opened, the secondary RIB for the file is displayed and control is transferred to level three. If no file is opened, control is transferred to level four.

name/ext - The named file is opened on the current drive, or any drive if non-specific drive mode is enabled. The primary RIB for the file is displayed and control is transferred to level three.

pfn - The file indicated by the octal physical file number given is opened on the current drive. The primary RIB for the file is displayed and control transfers to level three.

I - The current physical file number is incremented and the new file thus indicated is opened. If no file corresponding to that physical file number exists on the current drive, the PFN is

incremented repeatedly until a file corresponding to the PFN is found. The primary RIB for the file is displayed and control is transferred to level three.

D - D works just like the I command above except that instead of incrementing the PFN, it is decremented.

#pfn - The directory sector containing the entry corresponding to the file indicated by the specified physical file number is displayed; then control is transferred to level five. Since only the last four bits of the PFN are relevant, the pfn specifier is equivalent to a relative directory sector number. These directory sector numbers are always specified in octal.

* - Return control to level one.

> - Show the second half of the current sector.

< - Show the first half of the current sector.

21.6 Level Three Commands To DUMP

When the cursor indicates that DUMP is functioning at level three, the LRN level, the following commands are accepted.

<enter> - The current sector is shown and control is transferred to level four.

number - Access and display the record indicated by the LRN specified. If the number given has a leading zero, it is assumed to be octal; otherwise it is assumed to be decimal. The number specified is the user (as opposed to system) LRN. The system LRN, the value in bytes one and two in the sector, is always two greater than the user LRN. The two numbers displayed at level three in the control line are the user LRN in decimal (the one with leading zeros suppressed) and octal (the one in parentheses, with leading zeros).

Pnnnnn - Print the next 'nnnnn' sectors to the printer selected on the command line.

I - Increment the current logical record number, access it and display the sector.

D - Decrement the current logical record number, access it and display the sector.

* - Return to the File level of control (level two).

> - Show the second half of the current sector.

< - Show the first half of the current sector.

21.7 Level Four Commands To DUMP

Level four of the DUMP command requires more detailed understanding of DOS physical disk addresses, and as such is not usually as useful as the LRN level. However, when access to a specific sector on the disk is desired, it can be achieved using DUMP level four. It is important to realize that the physical disk addresses specified are logical physical disk addresses, that is, the same format as is given to the DR$ and DW$ routines in the DOS. They are not necessarily the same as actual physical locations on the disk. For example, with DOS.C for the 9380 series diskettes, the logical disk addresses are remapped onto the diskette into different hard physical sector numbers than those indicated by the logical physical disk address. The important thing to understand here is that the disk addresses used in the level four control of DUMP are those that would be used to parameterize DR$ and DW$.

The commands accepted at level four of DUMP are as follows.

msb, lsb - Access and display the sector indicated at the given physical disk address on the current logical drive. The first field (most significant byte) is assumed to be in decimal unless a leading zero is supplied. The second field (least significant byte) is always considered to be in octal, regardless of whether a leading zero is supplied or not. The second field is separated from the first by a comma. The physical disk address given by the user is assumed to be valid. If it is not of the proper format, undefined results may occur. Users who are not sure of their understanding of DOS internal physical disk addresses should not use level four of DUMP.

Pnnnnn - Print the next 'nnnnn' sectors to the printer selected on the command line.

I - Increment the current physical disk address, access it and display the sector.

D - Decrement the current physical disk address, access it and display the sector.

* - Return control to level two if no file is opened, or level three otherwise.

> - Show the second half of the current sector.

< - Show the first half of the current sector.

21.8 Level Five Commands to DUMP

When the flashing cursor indicates that the DUMP command is operating at control level five (system directory sector level), the following commands are accepted:

number - Show the directory sector indicated by the low order four bits of the number specified. Since only the low order four bits of the number are used, it is not an error to specify simply the physical file number (PFN) of the file whose directory entry is to be examined. A leading zero indicates the number is in octal, otherwise decimal is assumed.

Pnnnnn - Print the next 'nnnnn' sectors to the printer seclected on the command line.

I - The current directory sector number is incremented and the corresponding directory sector is displayed.

D - The current directory sector number is decremented and the corresponding directory sector is displayed.

* - Return control to level two.

> - Show the second half of the current directory sector.

< - Show the first half of the current directory sector.

21.9 Error Messages

Only one error message is issued by the DUMP command. It is:

ERROR IN DOS FUNCTION. DUMP ABORTED.

If this error message occurs, it means that the DOS FUNCTIONs are probably incorrect on the disk, generally indicating that the disk in the booted drive has not been completely (or correctly) DOSGENed. If this is the case, SYSTEM7/SYS should be loaded using the latest copy of DOS as distributed by Datapoint.

CHAPTER 22. THE DUMP93X0 COMMAND

22.1 Purpose

DUMP93X0 represents one of three programs: DUMP9350, DUMP9370, DUMP9380. Each program functions on only one of the Datapoint type disks, 9350 series, 9370 series, or 9380 series respectively. In the following chapter, characteristics of a particular program or disk will be indicated by the specific drive type. Features common to all programs will be indicated by reference to "DUMP93X0", so the "X" can be at any time read as "5", "7", or "8". The examples that follow are primarily set for DUMP9370 use, since the 9370 disk uses the most complex address format. In general, the examples apply equally well to 9350 or 9380 disks, ignoring the head address used in the 9370 command. The DUMP93X0 command enables the programmer to inspect, record, or load physical disk sectors. DUMP93X0 is intended to be used only for extremely low-level disk examination by trained systems personnel. Most users will find the facilities provided by the DUMP command to be more useful for general disk examination purposes.

22.2 Use

DUMP93X0 can be invoked from an active DOS by keying in at the system console:

```
DUMP93X0
```

Since DUMP93X0 is a completely self-contained program, it can be run from an LGO cassette tape (unlike most DOS commands which rely on one or more of the DOS routines for their execution). In this mode, DUMP93X0 can occasionally be useful in helping to determine the problem when the DOS will not boot up from some disk. If a user intends to use DUMP93X0 in this way, he should take care to make an LGO tape and store it safely away somewhere, before he needs it.

DUMP93X0 can output physical disk records (sectors) to a local printer, the cassette deck, or to the screen, and can load sectors to disk from the cassette deck.

There are two command handlers in DUMP93X0. The primary command handler controls all DUMP93X0 functions except the screen dump. The screen dump requires its own syntax because it is an interactive, and more flexible, facility.

All commands to DUMP93X0 employ the same conceptual structure, though elements of commands may be implicit as well as explicit. The full explicit format for commands is:

```
DUMP9370:    Z AAA,BBB,CCC  DDD,EEE,FFF
DUMP9350:    Z AAA,CCC  DDD,FFF
DUMP9380:    Z AAA,CCC  DDD,FFF
```

where
- Z is the command
- AAA is the starting cylinder number
- BBB is the starting head on cylinder AAA(DUMP9370 only)
- CCC is the starting sector on that track
- DDD is the ending cylinder number
- EEE is the ending head on cylinder DDD(DUMP9370 only)
- FFF is the ending sector on that track

Notice that all disk addresses are "hard" physical disk addresses, as opposed to DOS standard-format (or "logical") physical disk addresses. All numbers input to DUMP93X0 are octal. Consult the appropriate appendix for a description of the physical addressing of the type of disk in use.

The command codes of the primary command handler are:

- P Print on the local printer
- S Screen dump
- CD Cassette dump
- CL Cassette load
- # Jump to DOS DEBUG
- * Return to DOS command interpreter
- A ASCII mode (for printer or screen dump)
- E EBCDIC mode (for printer or screen dump) (DUMP9380 only)
- O Octal mode (for printer or screen dump)
- @ Physical drive number

The command codes of the screen dump command handler are:

- * Return to the primary command handler
- # Jump to DOS DEBUG
- I Increment the (cylinder,head,sector) address
- D Decrement the (cylinder,head,sector) address
- C Cylinder address mode
- H Head address mode (9370 only)

S Sector address mode
A ASCII display mode
E EBCDIC display mode (9380 only)
O Octal display mode

The following operating instructions discuss the commands and their applications, with some examples, in more detail.

22.3 The primary command handler

As soon as DUMP93X0 has fully loaded, it displays its signon message on the screen. When the cursor appears at the lower left corner of the screen the primary command handler is ready to accept commands.

22.4 Using

DUMP93X0 with a Local Printer P - Print on the local printer

DUMP93X0 will print only to a 132 column local printer, address 0303. The 256 byte disk records (sectors) are listed 32 bytes per line, 8 lines per sector. Preceding each 8 line block of print is a short line giving the physical disk address of the printed sector. One sector or the entire disk may be dumped to the printer by a P command. After the last sector is printed the page is ejected to top of the next page.

Unless otherwise specified, the bytes are printed in octal, with a space separating each byte, except every eighth byte is delimited by a period. If the DUMP93X0 command is in the ASCII mode (set with the A command) characters that are valid ASCII characters will be printed in ASCII. Lower-case ASCII alphabetic characters are indicated by a preceding underscore (_). If the DUMP9380 command is in the EBCDIC mode, bytes that are valid EBCDIC characters will be printed in EBCDIC, lower case characters preceded by an underscore.

COMMAND EXAMPLES:

P 000,000,000 000,000,000

would dump to the printer the disk records from cylinder 000, head 000, sector 000, thru cylinder 000, head 000, sector 000. In other words, print only the one sector with the disk address 000,000,000.

Note from the following examples that the parameter fetching subroutine will make certain assumptions about information not explicitly given.

P 0,0,0 0,23,27

would dump to the printer the disk records from cylinder 000, head 000, sector 000, thru cylinder 000, head 023, sector 027. In other words, dump to the printer all of the sectors on cylinder zero. Note that it is not necessary to supply leading zeros in an address.

For 9350 series disks, the equivalent command, dump all of cylinder 0, is

P 0,0 0,67

For 9380 series disks, the equivalent command is

P 0,0 0,14

P 0 0

would do exactly the same thing as the previous example. When only the first number is given between spaces, it is taken to be a cylinder address, with a sector and head address of 000 assumed for the beginning cylinder. For 9370 disks, a head address of 023 and a sector address of 027 are assumed for the ending cylinder address. For 9350 disks, a sector address of 067 is assumed for the ending address. For 9380 disks, a sector address of 014 is assumed for the ending address.

P 4

would dump to the printer the disk records from cylinder 004, head 000, sector 000, thru cylinder 004, head 023, sector 027. In other words, all of the sectors on cylinder 4. When only one cylinder address is given, it is taken to be both the beginning and ending cylinder address. For 9350 series, the command would dump from cylinder 004, sector 000, through 004, sector 067. For 9380 series, the command would dump from cylinder 004, sector 000, through cylinder 004, sector 014.

P 67 70,7

would be assumed to mean: P 067,000,000 070,007,027 ,

or for 9350's P 067,000 070,007

or for 9380's P 067,000 070,007

22.5 Screen Display format

S - Screen dump

DUMP93X0 can display on the CRT one disk physical record (sector) at a time, in octal or ASCII (or EBCDIC for 9380). The address of the sector displayed is controlled in a manner analogous to the display of bytes in memory by the DOS debugging facility.

A special display format is utilized to enable all 256 bytes of a sector to be displayed on the screen at one time. Below is a diagram of what a screen dump of a sector would look like; given the CYL,HED,SEC address = 44,0,6 and each byte in the example sector is its location within the sector; (i.e., starting at the beginning of the sector, the bytes are in octal) 000, 001, 002, 003,. . ., 0377:

Illustration 4.4

Note from the diagram that:

The displayed sector address is in the upper left-hand corner of the screen. For 9350 disks, the cylinder and sector address is shown. For 9370 disks, the cylinder, head, and sector address is shown. For 9380 disks, the cylinder, physical sector, and logical sector address is shown. Each portion of the address is on one line; stated sequence above is top to bottom.

Each group of 10(octal) bytes is displayed in a contiguous block of digits.

Each block of 100(octal) bytes begins at the left side of the screen, preceded by an underscore (_).

Each block of 100(octal) bytes consists of 10(octal) groups of 10(octal) contiguous bytes; 3, 3, and 2 groups to a screen line, for the three lines required to display 100(octal) bytes.

The screen displays 400(octal) bytes, which is one disk

sector, 256(decimal) bytes.

To further break down the screen and enable quick location and reading of individual bytes, the first digit of every second byte is flashed on and off. Thus, each group of eight bytes is divided into four units of two bytes.

COMMAND EXAMPLES:

S 044,014,006

would mean: display cylinder 44, head 014, sector 6 on the screen. This command can only be given to the primary command handler, and after it is executed DUMP93X0 will be under the control of the screen dump command handler.

22.6 The Screen Dump Command Handler

Note that as in the DOS debugging facility, the command codes entered are not displayed, the command is merely immediately executed.

* Return to the primary command handler. The screen will be rolled up, the cursor turned on, and keyed commands will be displayed as they are entered at the lower left corner of the screen.
NOTE that the SHIFT key must be depressed at the same time as the asterisk (*) key.

\# Jump to the DOS debugging facility. # will not work if DUMP93X0 was loaded from an LGO tape.
NOTE that the SHIFT key must be depressed at the same time as the pound sign (#) key.

I Increment the cylinder, head, or sector address and display the sector at the new address. The new disk address will be displayed at the top left corner of the screen.

If the C (Cylinder address mode) command is in force when an I command is given, the cylinder address will be incremented by one, the head and sector addresses will not change. Cylinder address wrap-around occurs at 0312->000 (0114->000 for DUMP9380). Incrementing by cylinder address is useful for scanning quickly through a large file by steps of 4 (9380) or 8 (9350,9370) clusters per increment.

If the H (Head address mode) command is in force when an I

command is given, the head number will be incremented by one. If the head address was 023, it will wrap around to head zero and the cylinder address will be incremented by one. Note that the head address will increment across both the two logical packs on the physical drive. H is operative only under DUMP9370.

If the S (Sector address mode) command is in force when an I command is given, the sector address will be incremented by one. If the sector was the last on the track (014 for 9380, 067 for 9350, 027 for 9370), then the head or cylinder address, or both, are incremented by one and the sector address is set to zero. If the cylinder address was the last on the disk, it will be set to zero. Incrementing by sector enables scanning sector by sector through a file and inspection of the exact data on each disk record. Files which span logical cylinders or are non-contiguous on the disk (which includes most large files) will require more detailed understanding by the user of the DOS file structure (in order to avoid incrementing out of the file's allocated space) and are usually better examined using the DUMP command.

D Decrement the cylinder, head, or sector address and display the sector at the new address. Except for the direction of address change, the D command is functionally like the I command.

C Cylinder address mode. This command causes subsequent I or D commands to alter the cylinder address. Optionally, a cylinder address may be keyed in before striking the C key; the current cylinder address will be replaced by the entered value before the disk record is read and displayed. The entered digits will be displayed at the lower left corner of the screen. Note that the address must be an octal address. If more than three digits are entered DUMP93X0 will BEEP and the procedure must be restarted. If the address entered is not a valid cylinder address (for example, greater than 0312) the C command will be in force but the cylinder address will not be changed. Also note that only the eight least significant bits of the value entered will be taken for the address (an entered value of 444 would be interpreted as 044).

H Head address mode. This command causes subsequent I or D commands to alter the head number. Except for the fact that the H command modifies head addresses and sets head mode, it is similar to the C command. (DUMP9370 only.)

S Sector address mode. This command causes subsequent I or D

commands to alter the sector address. Optionally, a sector address may be keyed in before striking the S key. The address option is functionally similar to the C command. Sector address mode is the assumed mode of operation when the program is started.

A ASCII display mode. This command causes the bytes to be displayed in ASCII instead of OCTAL on the screen, for all bytes that have valid ASCII bit configurations. This is useful for examining text files on disk. Note that the ASCII mode will carry over to the P (print) command of the primary command handler unless changed by a subsequent O command.

E EBCDIC display mode (9380 only). This command causes the bytes to be displayed in EBCDIC instead of OCTAL on the screen, for all bytes that have valid EBCDIC bit configurations. This is useful for examining the index track (track zero) on a diskette, and for text files on IBM formatted diskettes. While DUMP9380 is in EBCDIC mode, sector addresses used are taken as physical sector numers. During ASCII or Octal modes the addresses are taken as logical sector numbers and are re-mapped to take sector skewing and radius spiraling into account (see Appendix C).

O OCTAL display mode. This command causes the bytes to be displayed in OCTAL instead of ASCII. OCTAL mode is the assumed mode of operation when the program is started.

22.7 Cassette Operations

CD - Cassette Dump
CL - Cassette Load

DUMP93X0 can write to the front cassette deck the contents of specified disk sectors, and can read DUMP93X0 tapes from the front deck to load specified sectors.

COMMAND EXAMPLES:

```
CD 000,000,000 000,002,027
```

would mean: dump the sectors from cylinder 000, head 000, sector 000, thru cylinder 000, head 2, sector 027 to the cassette in the front deck. In other words, dump the first three tracks of the disk to cassette. The CD command will dump from one sector to 500 sectors (all that will fit on a cassette), in contiguous sectors. The disk addresses given (explicitly or implicitly) must be from

lesser to greater (e.g. CD 40,0,0 36,0,27 would be invalid because the second address is less than the first address). If any fault is found in the addresses given, the message:

PARAMETER ERROR

will be displayed and the machine will BEEP. Refer to the discussion of the P (print) command for examples of explicit and implicit addresses in commands. If the command is correct, the message:

FRONT DECK SCRATCH ?

will be displayed. A reply of "Y" will cause the cassette dump to proceed, while a reply of "X" will cause an exit to the primary command handler. Any other reply will cause the question to be repeated. When the front deck is ready, the cassette dump will rewind the tape and begin dumping the specified sectors to tape as individual 256-byte records. When all of the sectors have been written, the tape is rewound and checked sector by sector against the sectors on disk. If the tape data does not match the disk data exactly, the cassette dump will abort with the message:

TAPE/DISK VERIFY FAILURE

and exit to the primary command handler. If the tape is correct, it is rewound and control is returned to the primary command handler.

CL 0 ,2

means: load the disk sectors addressed 000,000,000 thru 000,02,027 from the front cassette. Not more than 500 sectors may be specified to be loaded from a cassette. The cassette load read routines expect to find records of exactly 256 bytes on the tape for at least as many records as there are sectors to be loaded. If a record that does not meet the specifications is encountered before the last sector has been loaded, the cassette load will abort with the message

BAD DUMP TAPE

and return control to the primary command handler. It is not necessary that the records on the tape be written to the same disk addresses as from which they were read. Therefore, the CD and CL commands provide a means of moving sectors from place to place on one disk, or from one disk to another.

WARNING: Loading these sectors does not affect the C.A.T. Directory, or RIBs on a disk. Therefore, if the sectors are not loaded carefully into a matching file, they will be unallocated, unreferenced and probably cause FORMAT errors if read.

It is not necessary that a CL read all of the records that may be on a cassette, only that there are at least as many records on the cassette as there are sectors to be loaded. When the specified sectors have been loaded, the tape is rewound and the tape records are re-read and matched against the loaded sectors on the disk. If the data on the tape does not match the data on the disk, the cassette load routine will abort with the message:

TAPE/DISK VERIFY FAILURE

and exit to the primary command handler. If everything is correct, the cassette load routine rewinds the front tape and returns control to the primary command handler.

22.8 Drive Numbers

When DUMP93X0 begins execution it assumes that it is to deal with the disk in drive zero. The @ command instructs DUMP93X0 to deal with the disk in the specified physical drive.

COMMAND EXAMPLE:

@ 1

would mean: succeeding commands will refer to the disk in physical drive 1. The @ 1 command will remain in force until another @ command addresses a different physical drive. Note that the address parameter for the @ command consists of one and only one digit.

22.9 Error Messages

Some of the error messages produced by DUMP93X0 and their meanings are explained below.

PARAMETER ERROR

Occurs if an invalid command or disk address is given to the primary command handler. Note that all disk addresses must be expressed in octal.

SO MUCH ?

Occurs if a command is given to dump more than 10 cylinders to the printer. Note that one cylinder will fill 32 printer pages (8 pages for 9350, 2 pages for 9380), and ten cylinders would represent a very large file. Respond "N" if you really don't want the printer to print out that many pages of paper. Otherwise, "Y" will cause the printing to proceed.

CASSETTE TOO SMALL

Occurs if a command is given to dump too many cylinders to cassette.

TAPE/DISK VERIFY FAILURE

Occurs during the tape-against-disk check phase of a cassette dump or cassette load if the data on the tape does not match exactly the data on disk. The tape is rewound and the dump or load should be retried.

BAD DUMP TAPE

Occurs if a tape record is read that does not conform to the DUMP93X0 tape record format. If it occurs during a cassette load, no data from the bad tape record is written to disk.

DISK NOT ON LINE

This message is self-explanatory.

DISK PROTECTED

Occurs if the disk is protected and a cassette load command is given. Nothing will be written to the disk as long as the READ ONLY indicator is on.

C.R.C. ERROR

Occurs if a hardware read or write error persists after three attempts to accomplish the read/write unless the read error occurs during a printer dump command (so that data on bad sectors can be hard-copy recorded and examined). If a C.R.C. error occurs during a printer dump, the machine will beep.

BEEP (Audio signal)

The machine will BEEP if an invalid command is entered from

the keyboard. Also see C.R.C. ERROR.

SEEK INCOMPLETE

(9370 only)

This occurs if the disk controller SEEK INCOMPLETE status bit is set. This bit is set if a cylinder seek operation does not finish within 100 milliseconds. When this occurs, it generally indicates a hardware malfunction.

COMMAND ERROR

(9350 only)

This occurs if the disk controller COMMAND ERROR status bit is set. The DUMP9350 program should be reloaded if this happens. If it happens again, something is wrong with the processor, the I/O bus, the disk controller, or the disk drive.

SECTOR NOT FOUND

(9370 only)

This occurs if the disk controller SECTOR NOT FOUND status bit is set. This usually occurs as a result of the formatting information on a disk (as written by INIT9370) being incomplete or incorrect, but could also indicate a software or hardware malfunction.

(9350 only)

Same as COMMAND ERROR.

(9380 only)

Occurs if the disk controller SECTOR NOT FOUND status bit is set. This usually occurs as a result of the formatting information on a disk being incomplete or incorrect, but could also indicate a software or hardware malfunction.

CHAPTER 23. EDIT

23.1 Purpose

EDIT is used to create and update source files on the disk. The editor enables the creation of files in a variety of data formats: text files, assembler code files, DATABUS source code files, or many user-designed data files.

This chapter describes the general operation of the editor. Basic commands to find, add and modify text lines are described. Intermediate commands permit short-cuts for experienced users and the advanced commands allow the user to create complex commands for repetitive editing tasks.

23.2 Use

The EDIT program is parameterized as follows (where items in square brackets are optional):

EDIT <f1>[,<f2>][,<f3>][;parameter list]

<f1> is the source file, <f2> is the scratch file and <f3> is the definition file. The source file <f1> is assumed to have an extension of 'TXT' if none is provided. If there is no file of the specified name, a new file is created after checking with the operator. If no scratch file <f2> is specified, a primary scratch file 'SCRATCH/TXT' and a secondary scratch file 'SCRATCH/XTX' are used. Since the second scratch file will be named <scratch file name> /XTX, the extension '/XTX' for <f2> is NOT allowed. The definition file <f3> is assumed to be EDIT/DEF unless otherwise specified. Both the scratch file name and the definition file name are stored in the configuration sector. These names will be used unless they are otherwise specified on the command line. Both the scratch and definition file names are displayed in the signon message.

23.3 Parameter List

A parameter list, indicated by the SEMI-COLON (;) following the file specifications, may be included. That list may include nine parameters which are order independent. The possible parameters are:

```
[;[margin][tab key][mode][shift][line][update]
  [keyclick][space-compression][non-verification]]
```

At the start of an EDIT, the values for these nine parameters are taken from the command line. Values for [margins], [tab key], [mode], [shift], [line], [key click] and [space-compression] not given on the command line are taken from the source file configuration sector, if there is one. The source file is updated unless otherwise specified on the command line and verification mode is assumed. Care should be exercised to be sure that not more than one mode, margin, and so on are specified on the command line or the desired value may not be selected.

When a file has been edited with EDIT, a special sector called the "configuration sector" is written at the beginning of the file. The sector contains the tabs, modes and special characters in effect when the EDIT was completed. These default values are used in place of any such parameters not specified on the command line.

If no command line parameter list is provided and the source file has no configuration sector, or an unrecognizable one then "Assembler" mode with space-compression, a margin at 75, no keyclick and the space bar for tabbing is assumed.

23.3.1 Margin Bell

A number in the parameter list is taken to be the margin designator; this causes the margin 'bell' to ring at the designated margin. The default margin is 75.

For example ";30" causes the bell to ring in column 30.

23.3.2 Tab Key Character

A tab key character encountered in the parameter list, i.e., non-alpha, non-numeric, non-colon, replaces the assumed tab key character. (SPACE in Assembler, DATABUS and Comment mode, SEMI-COLON in Text mode.) The tab character cannot be the same as any of the modify command separators or the continue character.

For example, ";^" causes the caret key (^) to replace the assumed character as the tab key.

23.3.3 Mode

A different set of assumptions is used if one of the 'mode' parameters is set. If no mode is listed or 'A' is typed, Assembler mode is used. DATABUS or DATAFORM (D) mode simply changes the tab stops.

Comment mode 'C' enables the same set of assumptions that 'A' does. This option is different from the 'C' mode available in the previous DOS General EDIT.

Text mode (T) sets no tabstops, does no shift inversion and enables the word wrap-around feature (see the glossary). The [shift] option 'S' and the [line] option 'L' are recognized only if the text mode 'T' is set (either on the command line or in the configuration file). To activate line truncation instead of word wrap-around in Text mode, enter 'L' in the parameter list. To enable shift key inversion (see glossary) in Text mode, enter the parameter 'S' in the list. Text mode is especially useful for generating SCRIBE input files.

See the glossary for complete definitions of the various modes.

23.3.4 Update

During editing, the source file is transferred into the scratch file as the text is updated. A second scratch file may be also used as the edit proceeds. When the edit is terminated, the physical source file is normally updated.

The 'ONE-PASS' parameter 'O' may be set in the parameter list. Then, at the completion of the edit, the scratch file contains the updated information and the source file is unchanged.

23.3.5 Key-click

If the 'K' parameter is set, a 'click' sounds each time a key is struck.

23.3.6 Space Compression

EDIT normally space-compresses (see Text File Format, in Appendix G.) If an 'E' appears as a parameter on the command line, spaces are "expanded", that is, NOT space-compressed. In expanded mode, EDIT reads in either space-compressed or non-space-compressed data but puts out only non-space-compressed records. To clear expanded mode, enter a 'G' for GEDIT format as a parameter on the command line. The space compression option (either 'E' or 'G') is stored with the file in the configuration sector, but may be changed by putting the desired option on the command line.

23.3.7 Non-verification

Four EDIT commands, :B, :E/, :E\ and :O, question the user:

```
SURE?
```

before executing. In certain special cases, for instance, when running under CHAIN, it is inconvenient to provide the 'Y'. The non-verification parameter on the command line allows the user to answer 'YES' in advance by placing a 'Y' on the command line. When 'Y' has been placed on the command line and the commands :B, :E/, :E\ or :O are entered, no verification is requested from the user.

23.4 Examples

To perform standard Assembler code editing, enter the command:

```
EDIT <source>
```

To edit a file for input to the text processor, SCRIBE, enter the command:

```
EDIT <source>;T
```

To also change the margin bell to ring at column 35 (e.g. for

labels) enter the command:

```
EDIT <source>;35T
```

These parameters set the bell at 35 and select the Text mode. Note that the parameters are not order dependent; therefore, the command:

```
EDIT <source>;T35
```

achieves the same results.

```
EDIT <source>;E
```

produces a file in which all spaces are written out (or non-space-compressed).

To write the edited text into a second file, without updating the original file, enter the command:

```
EDIT <source>,<new file>;OT
```

If the file is Assembler code instead of text, simply omit the 'T'; if DATABUS, replace 'T' by 'D'.

A second file, with the same name as <f1> but with a different extension, may be used as the scratch file by entering:

```
EDIT <f1>,/<extension>
```

Note that the extension /XTX is not allowed for the scratch file specification.

Once the initial command (and parameter list) has been entered, the DOS Editor sign-on message will appear on the screen followed with the file's configuration information (e.g. tabs, special characters, margin, keyclick, word wrap-around, shift inversion and space-compression) with the cursor left on the 'command line'. From this position data may be entered, lines may be fetched from the source file, or EDIT commands may be entered.

23.5 Data Entry and Retrieval

23.5.1 Data Entry

To enter text, simply type on the bottom or "command" line. When the ENTER key is pressed the screen rolls up one line. The command line is once again blank and the cursor is at the beginning of the command line, ready to accept more input.

When a SPACE is typed to the right of the margin bell column (except in column 79) and word wrap-around is enabled, the editor will automatically roll up the screen and begin a new line. If a non-space character is typed into the last column of the screen and word wrap-around is enabled, the last word on the line is removed and, after the screen is rolled up, that word is placed on the command line, where data entry may proceed.

The lines that have been rolled above the command lines are referred to as "screen lines".

When typing on a "screen line" (as the result of a command), the ENTER key causes the cursor to return to the command line. To continue data entry at the same screen area, the Pseudo-ENTER key may be used. This key (DEL) causes (in all but command mode), a new blank line to be inserted at that point on the screen so that data entry may proceed.

The BACKSPACE key erases the last character and moves the cursor back one position. The CANCEL key erases the line back to the previous tabstop, or back to the beginning of the line if no tabs are set.

Typing the tab key character causes the cursor to move to the next tab stop to the right. If there are no tab stops to the right of the cursor, the tab key character is accepted as a normal data character.

23.5.2 Multi-line Record Entry

A record in a disk file is a string of characters, perhaps including space compression characters, that is terminated with an octal 015. Normally the length of a record is no greater than 80 characters, including the octal 015, and corresponds to a screen line.

Multi-line records are records in a disk file that are greater than 79 characters long and must be continued to the next line when displayed on the screen.

To enter multi-line records, use the continue character. The assumed continue character is '&'. When this character is keyed in the first column, the continue indicator (a "greater than" (>) on a slow screen or a solid triangle on a high speed screen) is displayed. If the continue character is keyed in any other column, it is accepted as a normal data character.
NOTE: This character cannot be modified into, or taken out of a line ... it must be keyed in.

Note that although multi-line records are indicated on the screen, the editor treats them as if they were separate lines. The copy command, for instance, copies only the pointed line and NOT all of the lines in the mulit-line record. The only time EDIT recognizes lines as part of a multi-line record is during output.

Any line beginning with a continue indicator is joined to the previous line on output, forming records greater than 79 characters, if required.

When creating or modifying multi-line records, trailing spaces on a line often need to appear to force the line to be greater than 79 characters. In EDIT, trailing spaces are NOT deleted when they have been entered or read in from a file. However, when EDIT modifies, appends or copies a line, trailing spaces are deleted from that line as in the previous DOS Editor. Note that to update a multi-line record with trailing spaces the line must be re-entered. However, if the 'E' option is entered on the command line these trailing spaces will not be truncated by the ':M', ':C' or ':A' commands.

23.6 Data Retrieval

To fetch data from the source file, hold down the DISPLAY key and press the KEYBOARD key. As long as the two keys are depressed, data is fetched, displayed on the command line and rolled up the screen. If the end of file is reached, no more data is fetched and the machine beeps.

To fetch data backwards, hold down the KEYBOARD key and press the DISPLAY key. As long as the two keys are depressed, the screen rolls down and prior lines inserted on the top line. If the beginning of the file contained in memory is reached, the machine beeps, and no more data is fetched.

To fetch a single line, the shifted DEL key may be pressed. Using this key insures that only one input line is fetched. If keying on the command line, and a colon is not in the first column, a pseudo-enter anywhere on the command line will fetch another line.

All of available memory is used as a circular buffer. As the operator proceeds through the file, EDIT writes lines from the end of the buffer out to disk, maintaining a post-screen buffer so the user can roll backwards. Rolling backwards is restricted by the size of the post-screen buffer. Of course, if the file is small enough to fit completely in memory, there is no restriction. Sometimes the processor appears to hang while it is doing buffer management.

23.7 EDIT Command Format

The text appearing on the screen lines (that is, the lines above the command line) may be edited using a set of 'commands'. A 'pointer' (>) in the left hand column of the screen indicates the line which the command affects.

To move the pointer up, press the KEYBOARD key. To move the pointer down, press the DISPLAY key. The pointer wraps around from the top to the bottom and vice versa.

Commands allow the user to delete a single line (:D) or part of the screen (:SC and :SB), insert (:I) a new line between the current lines on the screen and modify (:M) parts of a line by replacing text or inserting new text. Commands are also available to search the file for specific text (:F and :L), for the end of the file (:EO or :E*), or for a particular line by number (:G).

An editor command is always preceeded by a COLON (:). To enter a command, type a colon in the first column of the command line with the appropriate command character(s) and any necessary parameters. The command characters may be upper or lower case. To force a line to be entered with a colon in the first column, start the line with two colons (the first one is discarded and the line shifted left).

23.8 Basic EDIT Commands

The following commands are a few basic editor commands. The user can get started without worrying about complex command forms. Remember that the 'pointer' on the screen indicates the line affected by the command.

23.8.1 Setting Tabs

:T - TAB set - this command enables the user to reset the tab stops during execution. The command causes a line of numbers to be displayed across the bottom of the screen.

The operator should space over to each position where a tabstop is desired and type any non-blank character. These tab stops are meaningful during data entry. A maximum of 20 tab stops may be set.

See the section on 'Changing Tabs' for more information on setting tabs.

23.8.2 Setting TEXT Mode

:X - TEXT - this command enables word wrap-around and disables shift key inversion and space insertion after leading periods, pluses, and asterisks. It automatically enters the tab set command (:T), so that tab stops may be cleared by the operator. The tab key character is not changed; therefore, the ":[tab key]" command must be used to set a new tab key character if one is desired.

See the section on 'Changing Modes and Options' for more information.

23.8.3 INSERTing a Line

:I - INSERT - Perform a line insert at the pointed line.

This command causes the lines from the bottom of the screen to the pointed line (but NOT including the pointed line) to roll down one line on the screen. A blank line is inserted below the pointed line, the pointer moves down to point to the blank line and the cursor is left at the beginning of the new blank line where data entry may proceed.

If word wrap-around is enabled and the end of the new line is reached during text entry, the new line (and all lines above it on the screen) roll up leaving a blank line containing the overflow from the previously INSERTed line. The cursor is left following the overflow awaiting continued data enty. When the ENTER key is pressed, the cursor returns to the command line and the new text, if any, remains on the screen in its new position.

If the ENTER key is hit in the first column of the blank INSERTed line, the screen rolls up to fill the null line, leaving the screen in its original form.

If the Pseudo-ENTER key is used instead of the ENTER key to terminate an INSERTed line, another INSERTion is performed and the cursor remains on the newly INSERTed blank line awaiting data entry of another line.

To make complex changes to a line already on the screen, the operator may INSERT a line immediately below the original and then retype the line with the changes. The original line may then be DELETEd.

The INSERT command may also be used in the form

:I <string>

This form will perform a new line insert as described above, place the specified <string> on the new line, and return to the command line immediately.

23.8.4 DELETEing a Line

:D - DELETE - delete the entire pointed line.

Blanks are written over the entire pointed line and the cursor is left on the now null line where new text may be entered. When the new line has been entered, it may be terminated with the ENTER key. This leaves the new line in the place of the DELETEd line, the pointer at the new line and the cursor returned to the command line.

If no replacement text is needed, pressing the ENTER key in the first column of the pointed line causes the screen to roll up one line from the bottom to fill the pointed vertical position with the line that was originally below it, leaves the pointer in the same vertical position and returns the cursor to the command line. At the end of the file (last screen of data), this action leaves a blank bottom screen line. This line is a "phantom" line with no real existence in the text file. It will not be written to the disk.

If a new line is written in the place of the DELETEd line and word wrap-around is enabled, the new line plus the screen lines above the new line roll up, as described in the "INSERTing a Line" section.

If the new line is terminated with a Pseudo-ENTER key, the cursor does not return to the command line. Instead, an INSERTion is made at the pointed line.

For other approaches to deleting lines, see the "Deleting Lines" section in the "Intermediate Commands" chapter.

23.8.5 COPYing a Line

:A - APPEND - copies the pointed line to the bottom of the screen, and rolls the screen up one line. The cursor returns to the command line and the pointer stays with the original pointed line by moving up one position when the screen rolls up. (Use :A* to keep the pointer in the same vertical position on an APPEND).

:C - COPY - deletes the pointed line, rolls up the screen and copies the pointed line to the bottom of the screen.

When the COPY command is entered, the pointed line is DELETEd and text may be entered in the now blank pointed line. As with the DELETE and INSERT commands, multiple lines may be inserted in

the screen lines when word-wrap is enabled or by terminating lines with the Pseudo-ENTER key.

When the new text entry is terminated with the ENTER key, the cursor returns to the command line. The original pointed line has been written following the line that was on the bottom of the screen when the COPY command was initiated. If one or more new line has been inserted during the COPY, the user must roll up the screen to view the moved line.

If no new text was entered, i.e., the ENTER key was entered as the first character on the new line, the screen rolls up to fill the null line and the pointer remains in the same vertical position. Since the screen has rolled up, the pointer is now pointing to the line following the first copied line so that a group of lines may be easily copied to another part of the screen.

23.8.6 MODIFYing a Line

:M [old text][command separator][new text] - MODIFY - a modify command allows the operator to:

1) replace [old text] by [new text],
2) insert [new text] after [old text]
3) or append (i.e., truncate and add) [new text] after [old text].

For instance:

:M [old text]<[new text]

replaces [old text] on the pointed line with [new text]. The command:

:M [old text]>[new text]

inserts [new text] immediately following [old text] on the pointed line. The command:

:M [old text]\[new text]

truncates the pointed line immediately following [old text] and then appends [new text].

If [old text] is not found in the pointed line, the machine beeps and returns to the command line without making any modification to the pointed line.

Modifications at the end of the file (on the last screen of data) can create "phantom" lines as described under "DELETEing a Line". For the many various forms of this command see the 'MODIFY Command' section.

23.8.7 LOCATEing a Line

:L - LOCATE next - typed exactly :L[ENTER], finds the next line of text. If positioned at the end of the file, the 'next' line is the first line of the file.

:L [old text] - LOCATE match - searches for a line containing embedded text matching [old text]. Leading spaces should be supplied if meaningful.

For additional approaches to locating a line, see the 'File Search Commands' section.

23.8.8 ENDing EDIT

:E* - EOF without display - searches for the end of the file and, when it is reached, displays the last screen of text.

:E - END - causes the remainder of the logical source file to be copied to the logical scratch file and then, if the logical scratch is not the physical input file, the scratch file is copied back to the source file.

The command line is left on the screen as long as the copy from source to scratch is in progress. It is erased during the final copy from scratch back to source.

Note: If EDIT is exited by any other means than one of the :E commands the format of the scratch files is not guaranteed. Also, if a system ERROR such as "File Space Full" is encountered while a :E is in progress, the format of the files is not guaranteed.

23.9 Intermediate Commands

Most of the following commands are expansions of the ones in the previous section. One additional concept introduced in this chapter is that of "fields". A field is a portion of the line between two consecutive tabs. Field one is between the left margin and the first tab, field two is between the first and second tab, and so on. Even though up to twenty tabs may be set, only the first nine fields may be referenced.

23.9.1 Changing Special Characters

:[tab key] - change the tab key character to any non-alpha, non-numeric, non-COLON, non-ENTER character typed after a leading colon on the command line.

:[old modify operator] [new modify operator] - change the old modify operator to the new one specified.

:[old continue character] [new continue character] - change the old continue character to be new one specified.

:CH - display the current special characters.

For instance, if a user wants to use "]" for the tab key, "=" for the modify replace operator, "-" for the modify insert operator, and "¦" for the modify append operator, the following commands are typed:

```
:]
:< =
:> -
:\ ¦
```

Then to check that they were properly changed, the user types:

```
:CH
```

which displays:

```
:CH TAB KEY: ] CONTINUE: & MODIFY REPLACE: =  INSERT: -  APPEND: ¦
```

None of the modify operators nor the continue character nor the tab key character may be the same character, and the special characters must all be non-alpha, non-numeric, non-colon, non-ENTER. A beep sounds if a character change command is invalid.

At the end of EDIT, the special characters and tabs are stored in the updated file. The next time that EDIT is used with the file, the same characters are used if not changed by the command line parameters. The tabs and special characters are displayed below the sign-on message.

23.9.2 Changing Tabs

:T - TAB set - enables the user to reset the tab stops during execution. The command causes a line of numbers to be displayed across the bottom of the screen.

The operator should space over to each position where a tabstop is desired and type any non-blank character. These tab stops are meaningful during data entry and for referencing fields (the portion of the line between consecutive tab stops). A maximum of 20 tab stops may be set. If the cancel key is depressed during selection it will cause the numbers on the screen to no longer be displayed, but the tab set command remains in operation until the enter key is depressed.

:T [nn][,nn]... - TAB set by column number - enables the user to reset the tab stops by column number. For instance, entering ":T 9,15,30" sets the tabs to columns 9, 15, and 30. A maximum of 20 tab stops may be set. Tab numbers must be in ascending order.

At the end of EDIT, the tab positions and special characters are stored in the updated file. The next time that the file is edited, the same tabs and special characters are used. They are displayed immediately below the sign-on message.

:T* TAB display - this command will display the current tab settings on the screen.

Below are commands for setting tabs to pre-determined default values.

:TA - Set Assembler tabs at columns 9, 15, and 30.

:TD - Set Datashare/Databus tabs at columns 10 and 20.

:TS - Set SNAP tabs at columns 11, 21 and 38.

:RH - RPG Header - sets tab stops for RPG header specification at columns 6 and 15.

:RF - RPG File - sets tab stops for RPG file description specification at columns 6, 15, 24, 33, 40, 54, 66 and 70.

:RE - RPG Extension - sets tab stops for RPG extension specification at columns 6, 11, 19, 27, 33, 36, 40, 46, 52 and 58.

:RL - RPG Line - sets tab stops for RPG line counter specification at columns 6, 15 and 20.

:RI - RPG Input - sets tab stops for RPG input specification at columns 6, 15, 21, 44, 53, 59 and 65.

:RC - RPG Calculation - sets tab stops for RPG calculation specification at columns 6, 18, 28, 33, 43, 49, 54 and 60.

:RO - RPG Output - sets tab stops for RPG output specification at columns 6, 15, 23, 32, 38, 40 and 45.

:RS - RPG Summary - sets tab stops for RPG summary specification at columns 6, 14 and 23.

23.9.3 Changing Modes and Options

:X - TEXT - enables word wrap-around and disables shift key inversion and space insertion after leading periods, pluses, and asterisks. It automatically enters the TAB set command (:T), so that tab stops may be cleared by the operator. The tab key character is not changed; therefore, the ":[tab key]" command must be used to set a new tab key character if one is desired.

:XI - Invert TEXT - enables shift key inversion and disables word wrap-around and enables space insertion after leading periods, pluses, and asterisks. It automatically enters the TAB set command so that tab stops may be reset by the operator.

:K - Keyclick - causes the machine to 'click' every time a key is struck.

:KI - Invert Keyclick - turns off the 'click' set above.

23.9.4 Deleting Lines

The user may delete the leading part of a line, the whole line, or multiple lines.

:D - DELETE - deletes the entire pointed line (See 'DELETEing Lines' section in the 'BASIC EDIT COMMANDS' section.

:D [old text] - DELETE through - deletes all characters from the left edge of the pointed line through (and including) the specified [old text]. The remaining characters are left justified and re-displayed.

The cursor returns automatically to the command line.

:D[#] [old text] - DELETE through field - deletes all characters from the left edge of the pointed line through (and including) the specified [old text] in the specified field. The remaining characters in that field only are left justified and re-displayed. All characters following the specified field are also deleted.

The cursor returns automatically to the command line.

:D[#][SPACE][ENTER] - DELETE through previous text - uses the previously defined [old text] to perform a DELETE through operation. A field number may be specified if desired, in which case a DELETE through field operation is performed using the previously defined [old text].

:D* - DELETE display - display the [old text] currently in use for DELETE. After the saved string is displayed, the cursor remains at the end of the display and the operator must press ENTER to proceed. No DELETE is actually performed. DELETE uses the same save area as LOCATE, FIND, and QUERY.

:SC - SCRATCH above - this command erases the lines from the top of the screen down to the pointed line, inclusive. The cursor and pointer are moved to the top line where data entry may proceed. Note, the execution of this command may cause the relative line number count, used by the :GA command to become inaccurate. Exiting and re-entering EDIT will correct this problem.

:SB - SCRATCH below - this command erases the lines from the pointed line to the bottom of the screen, inclusive. The cursor is left on the pointed line, where data entry may proceed.

As with the DELETE command, additional lines of text may be inserted with word wrap-around, if enabled, or by terminating each line with Pseudo-ENTER instead of ENTER.

23.9.5 MODIFY Command

The general form of the MODIFY command is:

:M[V][#] [old text][modify operator][new text]

where [V] is the VERIFY option, [#] is the optional field number, and [modify operator] is the command separator which defines the action of the command. Both [old text] and [new text] fields are optional. If [old text] is omitted, the command takes effect at the left-most edge of the pointed line (or at the left edge of the specified field). If the [new text] field is omitted, a null, or zero-length field is used to execute the modification. Note that the [old text] cannot include any of the modify operator characters; [new text] may contain one of those characters. If necessary, the modify operators can be changed as described above in 'Changing Special Characters' to avoid bad interpretations of modify commands.

The VERIFY option causes the cursor to blink at the first character to be modified. The user then has three responses. If he presses 'Y' for 'YES', the modification takes place. If he presses the 'ENTER' key, control returns to the command line. If he presses any other key, the modify command continues to search the line for another occurrence of [old text] to modify.

23.9.5.1 Line Modification

The following descriptions are of the line modification version of the MODIFY command.

:M[V] [old text][replace operator][new text] - MODIFY (replace) - replace the specified [old text] by the specified [new text]. The "less than" character (<) is the default command separator which indicates replacement. If [new text] field is omitted, the [old text] is simply deleted and the line compressed to the left.

For example to modify the text line:

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG'S BACK.

The command: ":M BROWN<RED" causes the line to be redisplayed like this:

THE QUICK RED FOX JUMPED OVER THE LAZY DOG'S BACK.

The command: ":M .< 1234 TIMES." to the above line generates a line like:

THE QUICK RED FOX JUMPED OVER THE LAZY DOG'S BACK 1234 TIMES.

If the replacement causes the line to become longer than 79 characters and word wrap-around is enabled, the trailing word is wrapped around and a new line is inserted containing the entire last word. If the [new text] is shorter than the [old text] it replaces, the line is shortened.

After the pointed line is redisplayed, the cursor is returned to the command line.

:M[V] [old text][insert operator][new text] - MODIFY (insert) - the command separator "greater than" (>) is the default character causing the [new text] to be inserted in the pointed line immediately after the [old text].

If the line becomes longer than 79 characters, and word wrap-around is not in effect, the trailing characters are truncated. If, however, word wrap-around is on, the last word(s) are inserted on a new line.

:M[V] [old text][append operator][new text] - MODIFY (append) - the "backslash" (\) is the default command separator causing everything in the pointed line past the [old text] to be replaced by the [new text].

As in all MODIFY commands, if the pointed line becomes longer than 79 characters, truncation occurs if word wrap-around is not enabled.

:M or :M[#] - MODIFY repeat - typed exactly :M[ENTER] or :M[#][ENTER], uses the [old text][sep][new text] from the last MODIFY command. This is useful when making the same change repeatedly. Note that the field number is not saved, and must therefore be supplied if necessary.

:MV or :MV[#] - MODIFY VERIFY repeat - typed exactly :MV[ENTER] or :MV[#][ENTER] is the same as the above command except that it invokes verification.

:M* - MODIFY display - display the expression entered for the last MODIFY. After the saved command is displayed, the cursor is left at the end of the display and the operator must press ENTER to proceed. No MODIFY is actually performed.

Unless the 'E' option, on the command line, has been set trailing spaces on the modified line are always deleted. This action may cause a multi-line record to be shortened.

23.9.5.2 Field Modification

In field modification mode, the MODIFY command acts only on a specific field and does not expand or contract the entire line but maintains the integrity of all fields before and after the affected field.

:M[V][#] [old text][modify operator][new text] - MODIFY field - where the pound sign [#] is a number from 1 to 9 designating the TAB field to be modified (or the starting point to search for matching [old text]). This command may be executed in any of the previous Modify forms.

Modification is performed within the specified field only. Thus, a replacement or append shorter than the original field is blank filled and subsequent fields will maintain their position and content. An insertion longer than the specified field is truncated (with the exception of the last field whenever word wrap-around is in effect).

For example, in assembler mode, field 1 is the label, field 2 is the op code, field 3 is the expression and field 4 is the comment.

```
LABEL  OP     EXP            COMMENT
```

the label may be deleted by the command:

```
:M1 \
```

and the resulting line becomes:

```
       OP     EXP            COMMENT
```

Or, the expression field (EXP) could be changed to EXP+1 without disturbing the comment field position by the command:

```
:M3 EXP>+1
```

which generates:

```
LABEL  OP     EXP+1          COMMENT
```

To add a comment to a line previously containing none or to replace an existing comment field, enter:

```
:M4 \[new comment]
```

Remember when using the repeat form of the MODIFY command that the modification applies to the entire line if the field number is omitted.

23.9.6 Line Splitting

:V [old text] - SPLIT - split the pointed line after the text matching [old text] and insert the remainder of the line past the matching text below the pointed line. The pointer remains in its original position. This is useful for INSERTing sentences in the midst of text without doing a group of cumbersome MODIFYs.

23.9.7 Line Concatenation

:W - CONCATENATE - append the line below the pointed line to the pointed line. The pointed line is assumed to have a trailing space if word wrap-around is in effect. This is useful in text mode where a MODIFY has caused words to wrap-around to the next line and the operator wishes to include them with the following line.

23.9.8 File Search Commands

Manual, operator controlled, searches may be performed by depressing the KEYBOARD and DISPLAY keys simultaneously to cause data to be fetched from the file (forward or backward depending which key is pressed first) and displayed on the screen. This continues until either key is released. The :EO command performs the same function automatically, i.e., it causes lines to be fetched and displayed until the end of the file is reached. Pressing the DISPLAY key stops the :EO command until it is released. To terminate an :EO command, press the KEYBOARD key. To fetch a single line use the Pseudo-ENTER key (DEL).

The :E- command works the same way as the :EO except that it

fetches lines backwards through the file in memory rolling the screen down.

To find the end of a file without displaying the entire file (since the display is time consuming) use the :E* command. This searches for the end of file and displays the last screen of data. For more information the :E command, see the section on 'Terminating EDIT'.

FINDs and LOCATEs search the file for a line containing specific text. When a line has been found, it is aligned with the pointer on the screen so that it may be operated on. Lines above and below the line are also displayed. FINDs and LOCATEs allow field specification, that is, if a field is specified, only lines with the specific text in that field are found.

A FIND or LOCATE wraps entirely around the file. If the requested text is not found, the last screen image when the FIND or LOCATE was executed is displayed and the machine beeps. A FIND or LOCATE may be aborted by pressing the KEYBOARD key.

The [old text] specified for a FIND command is saved. The saved [old text] may be redisplayed or used again. FIND, LOCATE, DELETE, and QUERY store the [old text] in the same save area.

:F [old text] - FIND match - the input file is searched for a line starting with the specified [old text]. Leading spaces in the file's lines will be ignored and need not be entered as part of [old text] unless meaningful. Note that this command should be typed exactly :F[SPACE][old text].

:F[SPACE] - FIND same match - if the FIND command is followed by exactly one space and the ENTER key, the previous FIND (DELETE, QUERY, or LOCATE) [old text] is used for this FIND. Several occurrences of the same text may be searched out in this manner.

:F[#] - FIND in field [#] - search for a field starting with the desired text. Field FINDs may be used in either of the command formats: :F[#] [old text] or :F[#][space].

:F* - FIND display - the asterisk (*) after the FIND command causes the [old text] of the previous FIND, DELETE, QUERY, or LOCATE command to be displayed. The cursor is left at the end of the display and the operator must press ENTER to proceed. No FIND is performed.

:L - LOCATE next - typed exactly :L[ENTER], finds the next line of text. If positioned at the end of the file, the screen is

cleared and the 'next' line is the first line of the file.

:L [old text] - LOCATE match - similar to FIND match except that the locate command searches for a line containing imbedded text matching [old text]. Leading spaces should be supplied if meaningful.

:L[space] - LOCATE same match - typed exactly :L[SPACE][ENTER], uses the [old text] specified by the previous LOCATE, DELETE, QUERY or FIND command to perform a search.

:L[#] - LOCATE in field [#] - locate desired string in specified field. Field LOCATES may be used in either of the above command formats: :L[#] [old text] or :L[#][space].

:L* - LOCATE display - display the [old text] entered for the previous LOCATE, DELETE, QUERY, or FIND command. As in the FIND display, the cursor is left at the end of the display and the operator must press ENTER to continue. No LOCATE is actually performed.

There are several variations on the GET command that allow the user to quickly jump forward or backward through a file or to a specific line by number.

:G - GET next screen - clears the screen and refills it with the next screen image.

:G- - GET prior screen - clears the screen and refills it with the prior screen image.

:Gnnnnn - GET nnnnn lines - fetches nnnnn (from 1 to 65,535) lines from the file or until the end of file. For example, ":G1" rolls the screen up one line.

:G-nnnnn - GET nnnnn lines backward - same as the above command except that it fetches backwards through the file in memory.

:GAnnnnn - GET Absolute [nnnnn]th line - position the file so that the [nnnnn]th line of the file (counting from the beginning of the file) is displayed on the bottom line of the screen.

:G* - GET display - this will display the line number (see :GAnnnnn command) of the pointed line. This is displayed on the bottom of the screen and will remain until a key is pressed.

23.9.9 BYPASS End of File

:B - BYPASS - fetch a line from the file, bypassing the end of file. This may be a true end of file or one caused by RECORD FORMAT errors, PARITY errors, or a RANGE TRAP (see the section on Recovery Procedures). Subsequent lines may then be fetched by the normal mechanisms. This command is intended as a recovery tool for use only if the file has been accidentally shortened or contains badly formatted records.

Before executing the BYPASS command EDIT asks the operator "SURE?", requiring a "Y" response for execution of the command.

Note: There are certain file conditions which :B cannot handle and/or correct.

23.9.10 Terminating EDIT

:E* - EOF without display - searches for the end of the file and, when it is reached, displays the last screen of text.

:EO - EOF with display - causes the data to be fetched and displayed on the screen continuously until end of file is reached. The operator may make the search pause by pressing the DISPLAY key.

:E- - Display to the beginning of file - works exactly as the :EO command above except that it fetches backward through the file in memory, rolling the screen down.

:E - END - the end command causes the remainder of the logical source file to be copied to the logical scratch file and then the scratch file is copied back to the source file.

The command line is left on the screen as long as the copy from source to scratch is in progress; it is erased during the final copy from scratch back to source. When the final copy is completed, control is returned to DOS.

:E/ - END/DEL - same as the END command, except the edited file is truncated below the last line on the screen.

More specifically, this command causes the remainder of the source file to be deleted (the lines currently on the screen are written out), and the scratch file is copied back to the source file. When the file is completely updated, the system is reloaded.

Before executing the END/DEL command EDIT asks "SURE?", requiring a "Y" response from the operator in order to execute the command.

:E\ - END/DEL - same as the END command, except the edited file is truncated above the first line on the screen.

More specifically, this command causes the prior portion of the source file to be deleted (the lines currently on the screen are written out), and the remainder of the file to be written out to the scratch file. The scratch file is then copied back to the source file. When the file is completely updated, the system is reloaded.

Before executing the END/DEL command EDIT asks "SURE?", requiring a "Y" response from the operator in order to execute the command.

:EX [DOS command string] - END and Execute - this command does an END (":E" above) and then executes the DOS command string. Note that if you have a program set for automatic execution (see the AUTO command), this program will be executed and the one set with the :EX command will be ignored.

:O - this command causes the EDIT to return to DOS without updating the source file.

Before executing END and Execute command EDIT asks "SURE?", requiring a "Y" response from the operator in order to execute the command.

:OX [DOS command string] - this command causes the EDIT to return to DOS without updating the source file and then execute the DOS command string.

If EDIT is exited using either :O or :OX the format of neither SCRATCH/TXT or SCRATCH/XTX is guaranteed correct. Also, if a file is created and EDIT is exited using :O or :OX rather than :E, then the format of the newly created file is not guaranteed. Note that in the case of an AUTO'd program, the AUTO'd program will be executed and not the one specified in the :OX command (see :EX also).

23.10 Advanced Commands

:0 [user-defined command string] - Define the user-defined command string zero.

:0 - Execute user-defined command string zero.

The user may define and execute up to ten EDIT command strings. These strings may use themselves, do conditional skips of commands, and request operator response. This gives the EDIT the capability of doing sophisticated file modification in a semi-automatic manner (the user always has complete control).

The user may define commands ":0" through ":9" as one or more EDIT commands. The defined command may be a single command (for example, a complicated MODIFY command) that is used quite often, but one the user does not want to type in every time he needs it. Or it may be a string of commands separated with Pseudo ENTER (DEL) characters. The Pseudo ENTER character appears as a solid triangle on a High-Speed or "RAM" screen and as a carat (^) on a slow screen. The command string should be terminated with a Pseudo ENTER if trailing spaces are significant. A carat is used to represent the pseudo ENTER in the examples below. For instance, if the user types:

```
:2 :M abcdefghijklmnop<ponmlkjihgfedcba
```

he has defined command 2. Every time that he types in:

```
:2
```

the pointed line is modified in the specified manner. If the user types:

```
:5 :M abcde<edcba^:M fghij<jihgf
```

he has defined command 5 as a pair of modify commands. Every time that he types in:

```
:5
```

the pointed line has both modifications done in sequence just as if the user had typed them in separately. When a command in a user-defined string fails, the string terminates, so if the first modifications fails the second one will not be performed.

If the user needs to replace every occurrence of the string "LABEL" in his file with "LABEL1", he may define a command as:

```
:2 :L LABEL ^M LABEL <LABEL1^2
```

Note: a new definition discards the old definition. Also, the colon following the Pseudo ENTER character is optional. The user may then type:

```
:2
```

which loops changing "LABEL " to "LABEL1" everywhere it is found in the file. The command string terminates when the LOCATE fails. Of course the user can always terminate the command with the KEYBOARD key.

For a more complicated example, the user may be editing the disk file which may be created by the DOS FILES Command, and define two commands as:

```
:8 :M2 \:DR0,:DR1^:M <COPY ^:9^
:9 :Z^:M  /</^:9^:G1^:Z^:Q /^:8^:Z*
```

Then he may set a tab at column 13 (immediately after the file extension in the FILES-created file). By typing:

```
:8
```

the user creates a CHAIN file for copying all the files from drive zero to drive one.

Modify commands used on the last screen of lines from the file generate a "phantom" line below the present screen lines. Creation of this line means that a :G1 command will always be successful following a modify. For this reason, repetitive commands should be constructed to use a locate or query command that will fail when the instruction should terminate. The examples above illustrate use of this type of test to terminate a repetitive command. The phantom lines that can appear at the end of a file during EDIT are null lines and will not appear in the updated disk file.

The execution of any command string may be temporarily stopped by holding down the DISPLAY key or terminated by pressing the KEYBOARD key.

:0* - Display the user-defined command string zero.

:00 - Display the user-defined command strings zero through nine.

The above commands allow the user to examine command strings individually in relation to the text on the screen or to examine them all at once in relation to each other.

:0< - Insert user-defined command string zero into the file text immediately below the pointed line.

:0> - Define user-defined command string zero as the pointed line on the screen.

The above commands allow the user to save the command strings in the text of his file. It also simplifies the modification of command strings as the user can use MODIFY on the string rather than keying in the entire string again. It also assists in defining several similar command strings.

A definition file with a default name of EDIT/DEF may be created by the user to contain a set of user pre-defined command strings. These are loaded automatically every time that EDIT is executed. This is an editable file. Remember, the user can force a colon as the first character on a line by starting the line with a double colon. The definition file may contain comment lines (lines starting with "+", "*", or ".") or null lines. The sequence of the defined command strings has no effect. Command strings may be multiply defined; the last definition will be the one in effect.

:99 [user-defined command string] - Define an initialization command string in the definition file.

If an initialization command is defined in the definition file, it is executed automatically when EDIT is executed. It may be defined to do things such as change the tab key, turn on key click, change the modify operators, set tab stops, or even do file modification without operater intervention. The automatic execution of the initialization command may be overridden by pressing the KEYBOARD key when executing EDIT.

:Z* - Terminate execution of the user-defined command string and return control to the bottom keyin line.

:Z - Skip over one command after the following command in the user-defined command string if the following command fails.

:Z[n] - Skip over [n] commands (0 to 9) after the following

command in the user-defined command string if the following command fails.

Almost every EDIT command either "succeeds" or "fails". For instance, LOCATE succeeds if it finds a line containing the specified text and fails if it doesn't. The MODIFY command fails if the string to be modified is not found or if, using the VERIFY option, the user has pressed the ENTER key. The GET fails if the end of file is reached. The use of the above commands allows conditionally skipping over commands in the user-defined command depending on the success of a command.

:U[n] - Unconditionally skip over [n] commands (1 to 9) in the user-defined command string.

This allows skipping over commands that might be jumped to by a conditional skip.

:Q [string] - QUERY, setting a succeed or fail condition, depending on the specified string being contained in the pointed line.

This command works similar to the FIND or LOCATE command in that it uses the line save area and allows field specification. It does not affect the pointed line at all but sets up a conditional skip. For instance, if command 3 were defined:

```
:3 :Z^Q2 ABC^Z*^G1^3
```

then its execution would GET lines until the pointer pointed to a line containing the string "ABC" in field 2. While the QUERY fails, the first conditional skip command (":Z") causes execution of the command string to skip over the ":Z*", GET a line, and then start over. When the QUERY succeeds, the ":Z*" is executed which terminates the command string. This example effectively does a LOCATE while displaying all the lines examined.

See the section 'Example of an EDIT/DEF File' for more user-defined commands.

23.11 Recovery Procedures

A 'FORMAT TRAP' occurs when a record not belonging to the current file is encountered. This can be caused either by a physical misalignment of the disk read head or because a record has erroneously been written into that file by some other program.

A 'RANGE TRAP' occurs when the physical limit of the file is reached and no end of file is present.

A 'PARITY TRAP' occurs when a record is misread from the disk. This may be caused by physical misalignment of the disk read head or a bad surface on the disk.

These three errors cause EDIT to believe that it has reached the end of file. To read past an end of file, use the BYPASS command, ":B", repeatedly if necessary.

Use of the B command may allow recovery of most of a damaged file, but at least some data will probably be irretrievable.

If the source file is lost (for example, erroneously KILLed), one of the scratch files may contain a useful copy. Since the scratch files (SCRATCH/TXT or SCRATCH/XTX) usually contain a copy of the last file edited, they may be used to recover only that file. If the file fits completely within the memory buffer, scratch files are never used.

23.12 Glossary

Assembler mode - assumed mode of execution. Tab stops at 9, 15 and 30. The space bar is assumed as the tab key character (this may be changed in parameter list or during execution). Shift key inversion and no word wrap-around are assumed. Leading periods (.), pluses (+), and asterisks (*) generate a following space (.) or (+) or (*) for comment lines.

Command - characters typed at the left edge of the command line following a COLON (:) which have special meaning to the editor.

Command line - the bottom line of the screen where most data is entered, lines are fetched and commands are typed.

Comment field - in assembler code the fourth field which is generally used for programmer comments.

Comment mode - assumed if 'C' in parameter list. Tab stops at 9, 15 and 30. The space bar is assumed to be the tab key character (this may be changed in parameter list or during execution). Shift key inversion and no word wrap-around are assumed. Leading periods (.), pluses (+), and asterisks (*) generate a following space (.) or (+) or (*) for comment lines.

Configuration sector - the first sector at a file which has been written by EDIT. It is invisible to most programs and therefore may be lost if the file is MOUTed, for example. When the configuration sector exists, it contains the tabs, bell margin, special characters, click option, mode and space-compression information and is used for the editor defaults.

Continue Character - a character which when entered in the first column causes a continue indicator to appear (see below). The default continue character is the ampersand (&) which may be changed during execution.

Continue Indicator - A solid triangle (or a "greater than" symbol (>) on a slow screen) appearing in the first column of a line, which means that the previous record exceeds 79 characters. When the continue character has been entered in the first column, the continue indicator appears and its presence means the line containing the indicator is joined on output to the previous line, possibly creating records greater than 79 characters.

DATABUS mode - assumed if 'D' in parameter list. Tab stops at 10 and 20 (may be changed during execution). The space bar is assumed to be the tab key character (this may be changed in the parameter list or during execution). Shift key inversion and no word wrap-around are assumed. Leading periods (.), pluses (+), and asterisks (*) generate a following space (.) or (+) or (*) for comment lines.

Definition file - this is an EDITable file containing pre-defined user command strings which is automatically loaded when the Editor is executed. The definition file may also contain an initialization command (":99") which is automatically executed unless the "KEYBOARD" key is pressed. The default name for the file is EDIT/DEF.

Field number - a digit used in commands to designate the portion of the pointed line between two consecutive tab stops. Field '1' is always from column 1 to the first tabstop; thus, in Assembler mode, '1' designates the label field, '2' the opcode field, '3' the expression field and '4' the comment field. During field modification, leading and trailing fields are preserved.

Line insert - results from an INSERT command, data entry or modification when word wrap-around is in effect, or use of the Pseudo-ENTER key. The lines below the pointed line are rolled down and a new, blank line is generated at the pointed line.

Logical scratch file - current output file.

Logical source file - current input file.

Modify operator - the character in a MODIFY command which indicates what is to be done. The default replace operator is the "less than" symbol (<), the default insert operator is the "greater than" symbol (>), and the default append operator is the "backslash" (\).

Multi-line Record - a line on disk that is greater than 79 characters, displaying as more than one line on the screen with each continued line marked with a continue indicator.

New text - a group of characters, typed immediately after a modify operator in a modify command, which will become part of the line being modified.

Old text - a group of characters, including spaces, which are searched for, either in the pointed line (as in the MODIFY command) or in the file (as in the FIND or LOCATE commands).

One-pass option - assumed if 'O' in parameter list. The one-pass option does not update the physical source file.

Parameter list - initialization information provided when the editor is first executed. Following file specifications, a SEMI-COLON (;) indicates the presence of a parameter list. The mode (A, C, D, or T), one-pass option (O), tab character, margin bell column, key-click (K), space-compression (E or G), non-verification (Y) and, in text mode only, 'shift inversion' (S), and 'no word

wrap-around' (L) may be set.

Pointed line - a pointer (>) in the left hand margin is used to reference lines for modification by command. The line to the right of the pointer is the pointed line.

Physical scratch file - specified (or implied SCRATCH/TXT) output file.

Physical source file - specified input file.

Pseudo-ENTER - the key marked DEL (always shifted) is referred to as the Pseudo-ENTER key. If pressed in the first column of the command line, one line of text is fetched from the source file.

If pressed while entering a command, a user-defined command string separator is entered.

In all other modes, the Pseudo-ENTER key causes a new line to be inserted so that data entry may proceed in the same area of the screen. If pressed on the last screen line, causes the processor to beep.

Scratch file - at any point in time, the logical scratch file is the output file.

Screen line - any of the lines on the screen which may be referenced by the command pointer. The command line (bottom line) is not, therefore, included.

Shift key inversion - reverse the function of the shift key for all alpha characters so that unshifted alpha characters appear upper case.

Source file - originally this is the input file specified at initial execution. The term source file refers to the current input file; thus, at any point in time, the logical source file may be either the specified input file or the scratch files.

Space-compression - an "abbreviation" written to disk file when two or more blank spaces are consecutive in a string. For example, "HI THERE" is converted to "HI(011)(005)THERE" where (011) is the space compression indicator, (005) is the number of spaces compressed and integers in parenthesis are octal codes. The default mode is 'G' for GEDIT or space-compressed mode. The 'E'

option on the command line causes spaces to be expanded, that is, written out with no space-compression.

Text mode - assumed by a 'T' in the parameter list. No tab stops are set (tabs may be set during execution). The SEMI-COLON (;) is the assumed tab character (the tab key character may be changed in the parameter list or during execution). No shift key inversion is performed (this may be selected in the parameter list with an 'S'). Word wrap-around is performed (this feature may be turned off by an 'L' in the parameter list).

Word - a word is defined as any group of less than 70 characters preceeded by a space.

Word wrap-around - a feature of text mode. During data entry a space to the right of the margin bell column causes an immediate carriage return. If this occurs on a screen line, a line insert is performed so that data entry may proceed at the same area of the screen. If a character is typed over the last column of the screen, the last word is removed, a line insert performed and the removed word is placed at the beginning of the inserted line where data entry may proceed. If a modify command causes the line to become longer than 79 characters, the trailing characters, including the last word on the line, are moved to a new line which is inserted below the original line. Control then returns to the command line.

23.13 Command List

The full set of EDIT commands are listed below. All legal combinations of options are included. [SP] represents a space. [ENT] represents the "ENTER" key. [#] represents the field number, a number in the range 1-9. Commands may be either upper or lower case.

:[NEW TAB KEY][ENT] - Replace the old tab key character with the new one. The default for the tab key character is the space bar.

:[OLD CONTINUE CHARACTER][SP][NEW CONTINUE CHARACTER][ENT] - Replace the old continue character (for input of long records) with the new one. The default for the continue character is the ampersand (&).

:[OLD REPLACE OPERATOR][SP][NEW REPLACE OPERATOR][ENT] - Replace the old modify-replace character with the new one. The default for the modify-replace character is the "less than" (<) symbol.

:[OLD INSERT OPERATOR][SP][NEW INSERT OPERATOR][ENT] - Replace the old modify-insert character with the new one. The default for the modify-insert character is the "greater than" (>) symbol.

:[OLD APPEND OPERATOR][SP][NEW APPEND OPERATOR][ENT] - Replace the old modify-append character with the new one. The default for the modify-append character is the "backslash" (\).

::[string][ENT] - Force a line beginning with a colon to be entered into the text of the file.

:A[ENT] - Copy the pointed line to the bottom of the screen and roll the screen up one line and move the pointer up one line.

:A*[ENT] - Copy the pointed line to the bottom of the screen and roll the screen up one line without moving the pointer. This allows defining a command that does multiple ":A*"s to copy multiple lines.

:B[ENTER] - BYPASS the end of file which may be caused by RECORD FORMAT, PARITY, or RANGE errors.

:C[ENT] - Copy the pointed line to the bottom of the screen, delete the pointed line, and key in a new line.

:CH[ENT] - Display the current tab key character and MODIFY operators.

:D*[ENT] - Display the previously defined string used by DELETE, FIND, LOCATE, or QUERY.

:D[ENT] - Delete the pointed line and key in a new line.

:D[SP][ENT] - Delete up through a previously defined string (defined by a previous DELETE, FIND, LOCATE, or QUERY) in the pointed line.

:D[SP][string][ENT] - Delete up through the specified string in the pointed line.

:D[#][SP][ENT] - Delete up through a previously defined string (defined by a previous DELETE, FIND, LOCATE, or QUERY) in the specified field of the pointed line.

:D[#][SP][string][ENT] - Delete up through the specified string in the specified field in the pointed line.

:E[ENT] - END the EDIT by copying the modified file back to the original source file.

:E/[ENT] - END the EDIT by copying the modified file back to the original source file up to the bottom line displayed on the screen. This deletes the remainder of the file.

:E\[ENT] - END the EDIT by copying the modified file back to the original source file starting with the top line of the screen up to the end of the file. This deletes the preceding portion of the file.

:EO[ENT] - Display the file forwards (rolling the screen up) to the end of file. The "DISPLAY" key stops the display until it is released.

:E-[ENT] - Display the file backwards (rolling the screen down) to the beginning of the file contained in memory. The "DISPLAY" key stops the display until it is released.

:E*[ENT] - Display the last screen of the file immediately above the COMMAND LINE.

:EX[SP][DOS command string][ENT] - END the EDIT by copying the modified file back to the original source file, then execute the DOS command string.

:F*[ENT] - Display the previously defined string used by FIND, DELETE, LOCATE, or QUERY.

:F[SP][ENT] - Find a line starting with the previously defined string.

:F[SP][string][ENT] - Find a line starting with the defined string.

:F[#][SP][ENT] - Find a line starting in the specified field with the previously defined string.

:F[#][SP][string][ENT] - Find a line starting in the specified field with the defined string.

:G[ENT] - Display the next screen-full of lines.

:G-[ENT] - Display the prior screen-full of lines.

:G[nnnnn][ENT] - Roll up the screen [nnnnn] lines where the number may range from 1 to 65,535 lines. This stops at the end of file.

:G-[nnnnn][ENT] - Roll down the screen [nnnnn] lines. This stops if the beginning of the memory buffer is reached.

:GA[nnnnn] - Display the [nnnnn]th line of the file. This rolls forward or backward through the file depending on the current location in the file.

:G* - Display the line number of the pointed line.

:I[ENT] - Insert by keying in a new line immediately below the pointed line.

:I[SP][string][ENT] - Insert the specified string immediately below the pointed line.

:K[ENT] - Turn on the key click.

:KI[ENT] - Turn off the key click (Key click Invert).

:L*[ENT] - Display the previously defined string used by LOCATE, DELETE, FIND, or QUERY.

:L[ENT] - Roll up the screen one line. At the end of the file, this causes the file to wrap-around to the beginning.

:L[SP][ENT] - LOCATE a line containing the previously defined string.

:L[SP][string][ENT] - LOCATE a line containing the defined string.

:L[#][SP][ENT] - LOCATE a line containing the previously defined string in the specified field.

:M*[ENT] - Display the previous MODIFY command line.

:M[ENT] - Modify the pointed line using the previous MODIFY command line.

:MV[ENT] - MODIFY with VERIFY the pointed line using the previous

MODIFY command line.

:MV[#][ENT] - Repeat the previous MODIFY command line with VERIFY applied to the specified field.

:M[SP][modify string][ENT] - MODIFY the pointed line as specified by the modify string.

:MV[SP][modify string][ENT] - MODIFY with VERIFY the pointed line as specified by the modify string.

:M[#][SP][modify string][ENT] - MODIFY the specified field in the pointed line as specified by the modify string.

:MV[#][SP][modify string][ENT] - MODIFY with VERIFY the specified field in the pointed line as specified by the modify string.

Modify strings are of the following formats. The default replace operator is the "less than" symbol (<). The default insert operator is the "greater than" symbol (>). The default append operator is the "backslash" symbol (\). [string1] and [string2] are optional.

[string1][replace operator][string2] - Replace string1 with string2.

[string1][insert operator][string2] - Insert string2 immediately following string1.

[string1][append operator][string2] - Truncate the line immediately following string1 and append string2.

:O[ENT] - Return to the Operating System without updating the original source file.

:OX[SP][DOS command string] - Return to the Operating System without updating the original source file and execute the given DOS command string.

:Q*[ENT] - Display the previously defined string used by QUERY, DELETE, FIND, or LOCATE.

:Q[SP][ENT] - QUERY (setting a succeed or fail condition) if the previously defined string is contained within the pointed line.

:Q[SP][string][ENT] - QUERY if the given string is contained

within the pointed line.

:Q[#][SP][ENT] - QUERY if the previously defined string is contained within the specified field of the pointed line.

:Q[#][SP][string][ENT] - QUERY if the given string is contained within the specified field of the pointed line.

:RC[ENT] - Set RPG Calculations tab stops (columns 6, 18, 28, 33, 43, 49, 54, and 60).

:RE[ENT] - Set RPG Extension tab stops (columns 6, 11, 19, 27, 33, 36, 40, 46, 52, and 58).

:RF[ENT] - Set RPG File tab stops (columns 6, 15, 24, 33, 40, 54, 66, and 70).

:RH[ENT] - Set RPG Header tab stops (columns 6 and 15).

:RI[ENT] - Set RPG Input tab stops (columns 6, 15, 21, 44, 53, 59, and 65).

:RL[ENT] - Set RPG Line tab stops (columns 6, 15, and 20).

:RO[ENT] - Set RPG Output tab stops (columns 6, 15, 23, 32, 38, 40, and 45).

:RS[ENT] - Set RPG Summary tab stops (columns 6, 14, and 23).

:SB[ENT] - SCRATCH BELOW deletes all the lines on the screen from the pointed line down through the bottom line and then allows keying in a new line at pointed line.

:SC[ENT] - SCRATCH deletes all the lines on the screen from the top line down through the pointed line and then allows keying in a new line at the top screen line.

:T[ENT] - Set TAB stops by displaying a line of column numbers and allowing the user to space across setting tabs by typing non-blanks. Up to 20 tabs may be set.

:T[SP][nn][,nn]...[ENT] - Set tab stops to the columns specified by [nn][,nn]... where [nn] ranges from 2 to 79. Up to 20 tabs may be set.

:T* - Display the current tab stop settings.

:TA[ENT] - Set TAB stops for Assembler (columns 9, 15, and 30).

:TD[ENT] - Set TAB stops for Databus or Datashare (columns 10 and 20).

:TS[ENT] - Set TAB stops for SNAP (columns 11, 21, and 38).

:U[ENT] - UNCONDITIONAL skip over the following command in the user-defined command string.

:U[n][ENT] - UNCONDITIONAL skip over [n] (1 to 9) following commands in the user-defined command string.

:V[SP][string][ENT] - Split the pointed line after the text in the line matching [string] and insert the remainder of the line immediately below the pointed line.

:W[ENT] - Concatenate the line below the pointed line to the pointed line. If word wrap-around is in effect, the pointed line is assumed to have a trailing space.

:X[ENT] - Change to TEXT mode with word wrap-around and no shift inversion and then set tab stops (as in :T above).

:XI[ENT] - Change to Assembler mode with shift inversion and no word wrap-around and then set tab stops (as in :T above).

:Z[ENT] - Skip over 1 command after the following command in the user-defined command string if the following command fails.

:Z[n][ENT] - Skip over [n] (0 to 9) commands after the following command in the user defined command string if the following command fails.

:Z*[ENT] - Terminate execution of the user-defined command string and return control to the bottom key in line.

The following commands, though refering to 0, actually refer to all the user-definable commands 0 through 9.

:0[ENT] - Execute user-defined command zero. Any value 0 through 9 may be used in place of zero.

:0*[ENT] - Display user-defined command zero. Any value 0 through 9 may be used in place of zero.

:0[SP][user-defined command string][ENT] - Define user-defined command zero. Any value 0 through 9 may be used in place of zero. The command string consists of one or more EDIT commands separated with a Pseudo ENTER (DEL) character. The colon immediately following the Pseudo ENTER character is optional.

:0<[ENT] - Insert user-defined command zero into the file text immediately below the pointed line. Any value 0 through 9 may be used in place of zero.

:0>[ENT] - Define user-defined command zero as the pointed line on the screen. The combination of this and the above command allow user-defined commands to be saved in the text of the file and to be edited. Any value 0 through 9 may be used in place of zero.

The above two commands, ":n<" and ":n>", cannot be used effectively from within a user-defined command string. Both commands function by moving the specified command string through the command string work area, so after they are executed, the command referenced in the instruction is present in the work area. If used within another user-defined command, the effect will be to terminate execution of the original string and begin execution of the string named in the :n< or :n> command.

:00[ENT] - Display the user-defined commands zero through nine.

:99[SP][user-defined command string][ENT] - This is the initialization command which appears in the definition file and is executed when the EDIT is started. It may be overridden by pressing the KEYBOARD key while bringing up EDIT. This command may only be used in the definition file (/DEF) and not during the actual EDIT.

23.14 EDIT ERROR MESSAGES

The following is a list of error messages that may occur during EDIT:

NAME REQUIRED - supply either the name of a new file or the name of the file that is to be edited.

BAD DEVICE - the drive specification on one of the three file specifications is either invalid or refers to a drive not on-line.

BOTH SOURCE AND SCRATCH FILES CANNOT BE SAME - either the same file specification has been entered in the first and second positions on the command line or the first file specification is "SCRATCH" and no scratch file specification is given (hence using the default name of SCRATCH).

BAD EXTENSION (/XTX) FOR SCRATCH - since EDIT uses two scratch files where the second scratch file has the name of the first scratch file and the extension XTX, the extension XTX is not allowed in the scratch file specification.

BAD OPTION PARAMETER - an invalid character appears on the command line.

INPUT FILE MUST EXIST IN "ONE-PASS" - the "O" option may not be used during creation of a new file

FAULTY DEFINITION FILE - either the file in the third file specification is not a valid definition file, or EDIT/DEF (the default definition file) does not contain the user-defined commands in the proper form.

TRAPS - FORMAT, RANGE and PARITY traps may occur when there is some problem with the head alignment of the disk drive or when there is something wrong with the file (see the chapter "Recovery Procedures")

Processor "beeps" - the usual indication of an error in the command just keyed in: rolling off the beginning of the text in memory, rolling off the end of the file, locating text not in the file, or keying in an unrecognizable command (see the "failure" conditions in the chapter "Advanced Commands").

23.15 Configuration Sector

A file that has been written with EDIT contains configuration information in the first sector. This sector begins with an octal 003 so that it is invisible to most programs. As a convenience, the tab settings, special characters and modes are contained in the sector so that these defaults are used the next time the file is edited. If the file has been MOUTed and MINed, for example, the configuration sector is not preserved.

The following describes the contents of the configuration sector. A three-digit integer in parentheses represents an octal byte.

(003) * 0 0 0 0 0 0 0 0 (015) E D I T <version> (015)

<tab numbers> (015) <bell position> (015)

<tab key> <continue character> <modify replace operator>

<modify insert operator> <modify append operator>

<key click switch> <shift inversion switch> <word wrap-around switch>

<text mode switch> <expand mode switch>

<scratch file name> <definition file name> (015) (003)

where:

<tab numbers>

are decimal integers separated by spaces representing the tab positions and the field lengths. For example, assembler tabs (9,15,30) are represented by "1 8 9 6 15 15 30 49 79".

<bell position>

is a decimal number of at most two digits.

<tab key>
<continue character>
<modify replace operator>
<modify insert operator>
<modify append operator>

are the actual ASCII characters.

<key click switch>

is "N" if off and "K" if the click option is on.

<shift inversion switch>

is "N" if no shift inversion and " " if shift inversion is on.

<word wrap-around switch>

is "W" for word-wrap and "N" for word-wrap off.

<text mode switch>

is "T" if text mode (spaces not automatically generated following a plus, asterisk or period in the first column) or "N" if not.

<expand mode switch>

is "E" when space compression is inhibited and "N" if space compression is desired.

<scratch file name>

is the eleven (11) character name and extension of the scratch file.

<definiton file name>

is the eleven (11) character name and extension of the definiton file.

23.16 Example of a Definition File

```
.  This is an example of a definition file for EDIT.  Note that comments
.  may appear.  Although this contains some useful general-purpose
.  user-defined commands, the most common use of command strings is for a
.  special editing job.  Care should be used in defining user command strin
.  to make sure they perform as intended!
.
.  For example, a file needs "?" following every line beginning with a "*"
.
```

```
:0 :t 2^:z1^:q1 *^:I ?^:g1^:0^
.
.  Many times user-defined commands are required for a one-time application.
.
.  Here are some more general examples:
.
.  Global search and replace with verification:
.
:1 :z^:mv^:1^:1 ^:1^
.
.  This function assumes that the modification and the locate have been
.  done previously so the repeated form of the command is used.  This
.  string can be modified to add field parameters for the locate and
.  modify commands or the verfication option removed from the modify
.  command.
.
.   Delete until:
.
:2 :z^:q ^:z*^:m \^:2^
.
.  By using the :Q function, "query" a particular string for the line the
.  cursor is pointing to.  This sets up a string for the repeated form of
.  the :Q command.  This function deletes all lines beginning with the
.  pointed line up until it finds the particular string entered at first.
.
.   Insert line of asterisks:
.
:3 :i *^:m *>**********^:m^:m^:m^:m^:m^:m^:m^:m^
.
.  Use this function to insert a line of asterisk (or modify the string
.  to insert lines of periods or underbars).
.
.   Draw a box of asterisks:
.
:4 :i *^:m *>**********^:m^:m^:m^:m^:m^:m^:m^:m^:t 79^:g-1^:5^
.
:5 :z^:q ***^:6^:m2 \*^:g-1^:5^
.
:6 :g1^:z^:q ***^:z*^:m1 <*  ^:6^
.
.  Use the line function (:3) to draw the top of the box.  Do a :xi to insure
.  that "format" (no word wrap) mode is in effect.  Now go down to where the
.  bottom of the  box is to be drawn and key in :4.  This draws a box of
.  asterisks.
.
.
.
.
```

```
.
.
.  Marking updated lines with SNAP tabs in assembler mode:
.
:7 :t 70^:m2 \ 1.1.D^:ts^
.
.  This is an easy way to mark modified lines.
.
. Leave :8 empty for "local" user-defined commands.
.
. End the file with a DOS command usually called after edit completion:
.
:9 :ex CHAIN ASSEMBLE/CHN;DATE=31OCT78,OPT=FIGLXMP,POPT,LINK=L^
.
. When the same chain file is usually called after the edit is finished, a
user-defined command may be used rather than typing in complicated paramet
each time.
.
```

Note: nested commands will not work. If the command is:

```
:1 :4^:6^:7^
```

when the user types in ':1', EDIT will perform Function 4, but will not return to perform Function 6 or Function 7.

Also if command :2 is defined as

```
:M *<:^:8>^:M :<*^:8
```

and the pointer is positioned to the line

```
"*EX SNAP3 PROG;LXIFG"
```

As soon as the first :8 is encountered

```
:EX SNAP3 PROG;LXIFG
```

will be placed in user defined position number 8, and user defined command :8 will be immediately executed. Execution control will not return to command :2.

Also, if command :3 is defined as

```
:8>^
```

and the pointer is positioned to the line

```
:EX EDIT PROG/TXT
```

EDIT will place in position :8

```
:EX EDIT PROG/TXT
```

and execute :8 immeditately

IF :3 is defined as

```
:8>
```

:3 will only place

```
:EX EDIT PROG/TXT
```

in position :8 and not execute it.

CHAPTER 24. ENCODE/DECODE COMMANDS

24.1 Purpose

The ENCODE command is used to convert/encrypt disk files containing data in any format into 79 character records containing only ASCII characters. Data in encoded format can be copied or transmitted by all Datapoint programs.

The DECODE command is used to translate encoded data files back into exact duplicates of the original disk files or decrypts encrypted files.

24.2 Use

ENCODE <file spec>,[<file spec>][;<key word>]

The ENCODE command converts the first file into encoded format and writes the data into the second file. If extensions are not supplied, ABS is assumed for the first file and ENC is assumed for the second file. If the second file is not specified, the name of the first file with an extension of ENC is assumed. The second file will be created if it does not already exist. Encoded data creates a file 50 percent larger than the original.

If the key word is given the file is encrypted using the key word as a basis. It is very important to retain the key word for the file cannot be retrived without it.

DECODE <file spec>,[<file spec>][;<key word>]

The DECODE command converts the first file from encoded format back into binary and writes the data into the second file. If extensions are not supplied, ENC is assumed for the first file and ABS is assumed for the second file. If the second file is not specified, the name of the first file with an extension of ABS is assumed. The second file will be created if it does not already exist.

If the file has been encrypted the key word must be given and the same as was used for ENCODE.

INPUT FILE MUST BE SPECIFIED!

will be displayed if the first file specification is omitted.

INPUT FILE DOES NOT EXIST!

will be displayed if the first file specified cannot be found in the DOS directory.

OUTPUT WOULD DESTROY INPUT FILE!

will be displayed if the first and second file specifications are identical.

INPUT FILE CONTAINS BAD DATA!

will be displayed if an encoded data file cannot be decoded into its original binary form.

This message also occurs when a file which has been encrypted is being decrypted with an incorrect key.

ENCODE reads and converts binary data until either a valid text end-of-file is read or allocated file space is exhausted. Data in encoded form is always terminated with a valid text end-of-file.

CHAPTER 25. FILES COMMAND

25.1 Purpose

FILES is a program which selectively prints or displays DOS file descriptions in file name sequence.

One may select information pertaining to all DOS files or to only those files with names and/or extensions beginning with the characters specified by the operator. Selected directory entries are sorted into ascending file name sequence. If desired, information from associated Retrieval Information Blocks (described in the chapter on System Structure) is also extracted for each directory entry. Extracted data is interpreted and displayed on the screen, listed on a Local or Servo printer, or written to a disk file.

25.2 Use

To execute the FILES program, type in the name FILES followed by selection criteria and display options (if option codes are to be used):

```
FILES [<filename>][/<ext>][:<drv>][,<subdir>][,<output-file>][;options]
```

<filename>	Select entries for files with names beginning with the 1-8 characters specified.
<ext>	Select entries for files with name extensions starting with the 1-3 characters specified.
<drv>	Specifies the disk drive to be selected. If this field is omitted, drive 0 will be selected.
<subdir>	Specifies the named subdirectory from which to select entries.
<output-file>	Specifies the disk file to which the selected entries will be written, if disk file output is specified.

options: The following option codes are available, and may be entered in any order:

N - Suppress file allocation map.
D - Display on CRT.
L - List on local printer.
S - List on servo printer.
F - Write output to disk as DOS text-type file.

If options are keyed and D, L, S and F are omitted, then D is assumed. D, L, S, and F options are mutually exclusive; output can be sent to only one device. If F is keyed and the <output file spec> is not present in the command line, one is requested by the message:

DOS OUTPUT FILE SPEC:

25.3 Default Messages

If no option codes are entered, the following messages will be displayed on the CRT:

SUPPRESS FILE ALLOCATION MAP?

If "Y" or "YES" is entered in response to this message, the display of file allocation information from Retrieval Information Blocks (RIB) will be suppressed. If any other response is entered, file allocation information will be displayed for each selected file.

After the user has replied to the map selection message, the program will test to see if the there is a servo printer connected to the processor. If a servo printer is attached and ready, the following message will be displayed:

LIST ON SERVO PRINTER?

If the user enters a "Y" or "YES" in response to this message, the servo printer will be selected to display output. If any other response is entered or the program cannot find an available servo printer, the program will test to see if a local printer is connected and ready for printing. If the program finds that a local printer is available, the following message will be displayed:

LIST ON LOCAL PRINTER?

If the user enters "Y" or "YES" in response to this message, the local printer will be selected for output. If a printer has been selected for output, the following message will be displayed:

ENTER HEADING:

Up to 32 characters can be entered, which will be displayed at the top of each page of printed output.

If no printer is available, or if the operator has rejected printer output, the program will ask for disk output:

WRITE OUTPUT ON DISK?

If the user enters "Y" or "YES", output will be written to a disk file, otherwise output will be displayed on the CRT. If disk output is selected, an output file name will be requested unless one was provided on the command line.

25.4 File Descriptions

File descriptions are sorted into ascending file name sequence for easy reference and displayed or printed in the following format:

FILENAME/EXT (PFN) DW

DW flags following the Physical File Number (PFN) indicate if the file is delete protected (D), or write protected (W). If the file allocation map was not suppressed, messages describing the file's size and location will be included in the file description. When allocation map information is printed or displayed, the program displays totals lines specifying the total number of files listed and the total number of sectors in those files. Disk output never has totals lines.

Depressing the DISPLAY key during display or printing of file descriptions will cause the program to pause until the key is released. Depressing the KEYBOARD key will cause the program to terminate and return control to the operating system.

Allocation map information describes each segment in the file by giving the cylinder and cluster starting address of the segment and its length in sectors. One line is displayed for each segment. See the chapter on System Structure and the Appendix for the appropriate DOS for a description of disk space allocation.

25.5 Error Messages

* PARITY ERROR *

FILES can not continue due to an irrecoverable parity error encountered while trying to read data from the disk.

* DRIVE OFFLINE *

FILES is unable to connect to the disk drive selected by the operator (drive 0 if not otherwise specified).

FILE(S) NOT FOUND.

No Directory entries have been found that meet the user' s selection criteria.

INVALID DRIVE

An invalid drive specification was entered.

CONFLICTING OPTIONS SPECIFIED

Options specify output on more than one device.

UNRECOGNIZABLE OPTION CODE

An unrecognizable code has been entered in the option field.

PRINTER NOT AVAILABLE

An option code specifies a printer that does not respond when tested for status.

CHAPTER 26. FIX COMMAND

26.1 Purpose

The FIX program can be used to modify bytes of DOS-loadable object code in an absolute code file. This program can be very dangerous and should be used only by qualified assembler language programmers or by someone following specific directions provided by Datapoint.

26.2 Use

To invoke FIX, enter the command:

FIX <file spec>

The program will display a sign-on message and will then display an initial line of six zeros, two spaces, and three more zeros on the bottom CRT line. (The zeros represent the current address and its contents.)

000000 000

The screen is then rolled up. The program then waits for a command from the operator. The <file spec> must specify a DOS-loadable object file. If no extension is provided, /ABS is assumed.

Commands are in the form [number][character] where the number is assumed to be octal. If the number is omitted, a value of zero is used. Commands are terminated by the enter key. Following a command, the current address and its contents are re-displayed.

26.3 Commands

The following is a list of command characters with their effect:

ENTER - Set current address.

- If no block of object code is currently in

memory (as at the beginning of execution or after a block has been rewritten), search the object file forward until a block containing the given location is found, then display the contents of that location. If the address does not exist in the object file, the current address is left at zero.

- If a block of code is in memory and the location given is within the limits of the block, the contents of the location will be displayed.

- If a block is in memory and the location given is not within the block limits, the current address will be set to the minimum or maximum address of that block, its contents will be displayed and a beep will sound. To access the desired address the current block must first be aborted (A) or transferred (T).

M - Change the contents of the current address to the number given.

I - Increment the current address (up to the maximum address in the current block).

. - Change contents of current address to number given and automatically increment the current address and display the contents of the resulting location.

D - Decrement the current address (down to the minimum address in the current block).

T - Transfer the modified block back to disk - rewriting it in place. After the block is written, the current address is set back to zero, so that all searches always start from the beginning of the file. No modification is made to the stored file until a T command is executed.

A - Abort processing the current block, set the current address back to zero.

O or * - Return to the operating system - if there is a block of object code in memory, it is <u>not</u> written back into the file.

If the command character is not one of the above, it is ignored and regarded as if only the ENTER KEY had been pressed.

26.4 Error Messages

If the <filespec> is not an absolute object code file, the message

RECORD FORMAT ERROR

is displayed.

If the file specified on the command line is not found, the message

NO SUCH NAME

is displayed.

CHAPTER 27. FIXAPPLY

27.1 Purpose

FIXAPPLY is a program which accepts as input patch files distributed by Datapoint Corporation for software maintenance purposes. Each encoded patch file contains the file names and member names of the modules to be patched, the patch addresses, patch data, and generated self-checking bytes to guarantee file integrity.

27.2 Use

FIXAPPLY is invoked using a command line of the form:

FIXAPPLY <patchfile>

where <patchfile> specifies the name of the patch file containing the modifications to be performed. All information needed for the modifications to be performed is included in the patch file, including the names of the files and library members involved.

FIXAPPLY execution consists of two phases: a verification phase and an execution phase. The verification phase reads the input patch file, locates all modules to be patched, verifies the prior data to be overstored, and guarantees the self-check fields to be correct. The execution phase reads a binary work file generated in the first phase and directly updates the object modules.

27.2.1 FIXAPPLY Phase One

The first phase of FIXAPPLY reads the patch file provided, converts the data into binary, validates it, and builds a work file on disk (FIXAPPLY/WRK) containing the random positions in the programs to be patched.

If any errors are found in the first phase, execution is terminated (with one exception, as described below). Note that the original value of the object byte to be patched is contained as part of the data in the patch file. This data must match,

preventing re-patching the same object module. It also allows version and revision levels to be checked.

Most errors occuring in phase one allow phase one to run to completion but will not allow phase two to run.

There is one phase one error which can be overridden by operator intervention at the system console. If FIXAPPLY fails to find a particular file, the message:

CAN'T FIND FILE <file name>

will be displayed. The operator will then be asked for the correct name of this file. This action allows users who have renamed Datapoint software products to specify their own in-house names. Member names within absolute libraries, however, cannot be overridden.

If fatal phase one errors are found, the message:

PHASE 1 ERRORS; ABORTED

will appear and the program will terminate, returning to DOS. If no fatal phase one error occurred, the message:

PHASE 1 COMPLETED OK

will appear and the program will proceed to its second phase.

27.2.2 FIXAPPLY Phase Two

There are no active displays during FIXAPPLY phase two, nor is there any means of operator intervention during this phase. It is important that the program not be interrupted (such as by rebooting the system) during this phase, as a partial upgrade of a target file could occur. At the end of phase two the message:

SUCCESSFUL COMPLETION

will appear and the program will return to DOS.

If some error does occur during phase 2 the message:

PHASE 2 INTERNAL ERROR; ABORTED

will appear and the program will immediately terminate and return to DOS. In the event of such a failure, the target file will be

partially updated and unusable and must be reloaded prior to again attempting to apply the patch. An internal error abort is probably caused by hardware failure, or modification of the target or work file by another user on a multi-user system.

27.2.3 Fatal Phase One Error Messages

INPUT FILE MISSING OR NOT SPECIFIED!
The patch file name was not specified or the specified file could not be found. The program terminates immediately without completing phase one.

OLD/NEW BYTE MISMATCH!
The object file is probably not the proper version/revision.

RECORD COUNT ERROR!
A record has been lost within the modifications for the current program name displayed on the screen.

ILLEGAL EOF ON INPUT!
The input file is not properly formatted.

INVALID OBJECT FORMAT!
The current object file is not in a recognizable format.

FILE INTEGRITY ERROR ON INPUT!
The internal LRC check has failed.

It is important to remember that the internal pseudo-LRC on the patch file prevents the file from being tampered with. Any alteration to the file will almost always cause pass one to fail. Also, due to the old/new byte checking, a patch file may not be applied twice to the same software.

CHAPTER 28. FREE COMMAND

28.1 Purpose

As a disk becomes full, it is useful to know how many 256-byte sectors remain available for allocation. Another useful bit of knowledge on the larger disks is how many empty slots in the directory remain for the allocation of file names. The FREE command displays these two values.

28.2 Use

The FREE command is invoked by a command line of the form:

FREE [:<drv>]

The drive to be examined may be specified by the <drv> field. If no drive is specified, all drives will be examined.

The command scans all drives that it finds on-line and displays (1) the number of available file names (representing possible files to be created) and (2) the number of available sectors that it finds on each.

Holding down the DISPLAY key will cause FREE to pause. Pressing the KEYBOARD key will cause FREE to terminate and return to the operating system.

CHAPTER 29. INDEX COMMAND

29.1 Purpose

The INDEX command creates or reorganizes the tree structure required by programs using the indexed sequential access method (ISAM). Indices may be created from any DOS text file. The indexed access method can then rapidly access records in this file in either sequential or random order. Records in files to be indexed must contain a record key up to 118 characters long contained in the first 249 characters of each record.

It is possible to build many independent indices to permit access to records of the same file by many separate, unrelated keys. There are no restrictions on the number of indices that may be built, or on the relationship or lack of relationship among the various keys used.

INDEX can create the tree structure (using the DOS SORT command), reorganize the tree structure, create a Keytag file from the index file, or create an index from a Keytag file. The format of a Keytag file is described in the chapter on the SORT command.

29.2 Use

When the Index command is to be executed, the operator must enter:

INDEX <textfile>[,<indxfile>][,<tagfile>][,<drive>];<parameters>

where only the first file specification and key field parameter are mandatory, and specify the text file to be indexed. Default extension is /TXT. The second file specification is the name of the INDEX file to be created. If no file is specified, the name of the first file is used with default extension of /ISI. If no drive is specified, the INDEX file will be placed on the same drive as the file to be indexed. INDEX files may have any names at all and may be located on physically different drives from the file being indexed. However, high-level languages using ISAM files (DATABUS, for example) assume the INDEX file will have the normal /ISI extension, and if the file open instruction is drive

directed the /ISI and /TXT files must be on the same drive.

The third file specification is for the intermediate tag file. The third file name will also default to the name of the first file with a default extension of /TAG. The fourth file specification, which may only specify drive, tells SORT where to put its intermediate work files. Otherwise, SORT will attempt to optimize drive selection.

29.2.1 Parameters

In addition to the parameters that INDEX itself recognizes, the user may specify any parameters acceptable to the REFORMAT utility (if preprocessing is to be done), or a primary record specification to be passed to SORT, or Mnnn or Q options to FASTSORT. Parameters recognized by INDEX are as follows:

```
        K -- Create a Keytag file from the /ISI file.
        I -- Create an /ISI file from the Keytag file.
        X -- Recreate the /ISI file, handling insertions and
             deletions.
        F -- Preprocess the input file with REFORMAT.
        E -- Index in EBCDIC collating sequence.
mmm-nnn -- Key specification
```

The Keytag file is a standard text file containing the pointer and key of each record to be indexed. The format is explained in the SORT chapter. The file may be LISTed, EDITed or transmitted. This last feature allows the /ISI file to be created at a remote site without invoking SORT.

The format of the key is mmm-nnn [,mmm-nnn] [,mmm-nnn]... where mmm is the beginning character position of the key field in each logical record and nnn is the ending position of the key field. Note that each record must have a unique key. Refer to the SORT chapter for a more complete description of key specifications.

The primary record specification is an option that allows the user to create the ISAM index file from a subset of the data file. The format of the primary record specification is "Pnnntc". The "P" must always appear. The field following "P", denoted by "nnn", represents the column in each logical record where a one position field exists that differentiates records in the file. The location of this one character field must be less than or equal to 249. The "t" can have one of two values: either an

equal sign (=) or a pound sign (#). If the former, it means create the ISAM index file from all records that contain the ASCII character "c" in position nnn. If it is a pound sign, it means that the ISAM file will be created from all records that do not contain the value of "c" in position nnn. The "c" may be any ASCII character except 015 (ENTER value).

In general the parameters for INDEX can be specified in any order and may optionally be separated from each other by a blank or a comma. The only exception to this rule is when a primary record specification exists, it must precede the key field specification and be separated from the key by a blank or a comma.

29.2.2 System Requirements

If INDEX is creating the index file from the text file (no K, I, or X option specified) the SORT or FASTSORT command must be available. If INDEX is being executed on a processor capable of executing FASTSORT, it will attempt to execute FASTSORT before defaulting to the SORT command. INDEX can locate FASTSORT in UTILITY/SYS or as a separate command file. FASTSORT is a separate program not released as part of DOS.

If pre-processing of the text file is specified on the INDEX command line, the REFORMAT command must be available to perform the pre-processing. INDEX can locate REFORMAT in UTILITY/SYS or as a separate command file.

If the X option of INDEX is used to recreate the tree structure file, the NAME command must be available. INDEX can locate NAME in UTILITY/SYS or as a separate command file.

If the E option of INDEX is used to specify EBCDIC collating sequence, the EBCDIC/SEQ file must be available to define that sequence.

29.3 Choosing A Record Key

Since the speed of access to an indexed file varies according to how much file space and thus how many levels of index are required for the index tree, the choice of what to use for a record key becomes highly important. Of course, you must choose a key which will uniquely determine the record you wish to access, but you should scrupulously avoid including information in the key which is not absolutely necessary. For example, a file could be

keyed according to automobile license plate numbers. Typically, these numbers will include a hyphen or other punctuation, which could easily be excluded from the record's key. The indexed access method will perform more efficiently if all non-significant characters are removed from the record's key.

29.4 Preprocessing the File

In file structures such as an indexed file where records are randomly inserted and deleted, the file tends to become non-optimum for searching. In addition, due to the method with which the indexed access method inserts records, each inserted record exists in a separate disk sector. This means that for records that are 80 characters long, two-thirds of the disk space for each additional record is wasted. This results in a reduction of the performance of the indexed access method.

In order to reclaim space vacated by deleted records and padding bytes in inserted records, the file may be processed by the REFORMAT utility prior to indexing.

29.4.1 Invoking Reformat

The INDEX utility will automatically invoke REFORMAT if the "F" option is present when INDEX is invoked. You must have specified the options that REFORMAT will need to process the file.

Note that if multiple indices are to be created, reformatting need only be specified for the first INDEX step, and MUST not be specified later if it was not specified in the first step. Although REFORMAT will not destroy the file, specifying reformatting may invalidate any previously built indices.

Basically, you must tell REFORMAT what format the records of the file are to have after preprocessing. You may select record compression, space and record compression, or blocking. Since the reformatting is done in-place, the REFORMAT option cannot enlarge the file which is to be indexed. For additional details on the REFORMAT utility, see the REFORMAT section of this guide.

29.4.2 Considerations for Unattended Indexing

Those who use the INDEX command from a CHAIN file (see the section on the CHAIN command for more details) and use AUTOKEY to restart their chain in the event of a failure should generally avoid using REFORMAT directly from INDEX. The reason why is that REFORMAT as invoked by INDEX uses the REFORMAT-in-place mode of the REFORMAT command. (The reason for this is that it is faster to do so, and also allows the indexing with reformatting of a file which is too big to REFORMAT in the available scratch space on a single-drive, almost full disk). Although REFORMAT is very careful not to damage the file being processed, if the file is actually in the process of being reformatted when a power failure occurs, the results can be undesirable.

This potential problem during unattended INDEX chaining can be avoided by setting a checkpoint (see the AUTOKEY command description for details), copying the original file to a scratch file, setting another checkpoint, reformatting the scratch file back into the original (using the COPY mode of REFORMAT), setting a further checkpoint, and finally INDEXing the file using INDEX. In this way there is always an undamaged file with which execution can resume if necessary.

29.5 INDEX Messages

The Index command displays several messages on the operator's console. They are listed below with explanations, in the sequence in which they may appear.

DOS. VER n.n INDEX COMMAND - date
: Signon message that gives the user the version of DOS required and the date of the INDEX command.

WRONG DOS!!
: Indicates that the version of DOS in use is too old to support the version of the INDEX command being used.

INFILE NAME MISSING.
: Indicates that the user has omitted the first, and required, file specification.

NO SUCH NAME.
: The input file specified for the INDEX does not exist.

INVALID DEVICE.

A drive specification entered as part of one of the file specifications was invalid.

SYSTEM7/SYS MISSING!
Indicates the SYSTEM7/SYS file is missing on the drive on which the /ISI file resides. This message only appears if the X option is used.

KEYTAG FILE BEING BUILT.
Indicates that INDEX is now creating the ASCII KEYTAG file requested with the "K" option.

FILE PREPROCESSING WILL BE DONE BY REFORMAT COMMAND.
This indicates that the user has requested preprocessing of his file by the REFORMAT command (R option specified).

INDEX WILL USE EBCDIC SORT.
The user has requested an index using the EBCDIC collating sequence (E option specified).

REFORMAT <filename>; <parameters>
Display of the command line used to invoke REFORMAT for preprocessing the input file.

REFORMAT UNLOADABLE!
Indicates that the REFORMAT command could not be found as a member of UTILITY/SYS or as REFORMAT/CMD, or that if found the command was not executable. The REFORMAT command should be loaded or reloaded onto disk.

SORT <filename>, . . .
Display of the command line used to invoke SORT to produce the Keytag file from which the index will be constructed.

SORT UNLOADABLE!
Indicates that the SORT/CMD file cound not be found, or that if found it was not executable. The SORT command should be loaded or reloaded onto disk.

BUILDING LOWEST LEVEL INDEX.
This indicates that INDEX is now creating the lowest level of the index file.

NULL INDEX FILE CREATED.
This indicates that an empty tag file was created by SORT. The index file created is usable by programs

using ISAM for adding records. After creating a null index file INDEX will exit normally (CHAIN will not be aborted) but will have set the ABTIF bit in DOSFLAG.

LONG KEY ENCOUNTERED AND TRUNCATED.
This indicates that the tag file contained a key that was longer than 118 characters. It was truncated to 118 characters.

DUPLICATE KEY: <key>
Two keys in the tag file were found to be identical and the first 60 characters of the key are displayed. INDEX will continue so as to display any other duplicate keys that may be found.

INDEX TERMINATED WITH DUPLICATE KEYS.
Duplicate keys have been found and a null index will be created, as described above under NULL INDEX FILE CREATED. The tag file is not deleted and since it is in standard text format, it may be EDITed to remove or modify the duplicate key and tag. Or a program (e.g. in DATABUS) may be written to display the records containing the duplicate keys so the user may resolve the ambiguity. INDEX may then be reinvoked using the "I" option.

BUILDING -NEXT- LEVEL INDEX.
This indicates that the lower level of the index file has been completed and the next level is now being created.

DONE.
The creation of the index file is now completed.

Other messages may be generated by REFORMAT or SORT. See the appropriate chapter for an explanation.

29.6 ISI File Formats

The DOS indexed file structure consists of a multi-level radix tree structure based on the record keys, and contains pointers to the location of the keyed records. Note that since many of these pointers are physical disk addresses, the ISI file cannot be moved without re-invoking INDEX. The text file may be moved so long as it is unchanged in any way. Moving the ISI file will destroy it.

The different levels of indices all have the same content, except for the lowest level index. Index levels are built up until an intermediate level of index will fit in a single disk sector. This becomes the highest level of index. This requirement is the reason for the 118 character limitation on key length.

The ISI files have the following format:

Offset	Length	Description
0	3	PFN and LRN bytes as per DOS convention - see the chapter on SYSTEM STRUCTURE.
3	nn	This is a KEY entry where nn is key length+7 for a lowest level index, and key length+3 for a higher level index. The first sector of an ISI file after the RIBs is a special header record.
nn+4	nn	This is the second KEY entry in the sector. There must be at least two KEY entries per sector. . . . Note that as many key entries are put in a sector as will fit without splitting across a sector boundary.

Each KEY entry for an intermediate level index has the following format:

Offset	Length	Description
0	KL	The highest key in the next lower level index sector. (KL is the keylength.)
KL	1	Octal 012 - This indicates the end of the key and that this is a higher level index entry.
KL+1	2	PDA (MSB, LSB) of the entry in the next lower level of index.

KL+3	1	Octal 0377 - This indicates that this is the last entry in this sector.

Each KEY entry for a lowest level index entry has the following format:

Offset	Length	Description
0	KL	The key for this particular record. (KL is the keylength.)
KL	1	Octal 015 - This indicates that this is a lowest level index entry and delimits the end of the key.
KL+1	3	Buffer Offset, and the physical disk address for the logically next lowest level index entry.
KL+4	3	Buffer Offset, and logical record number of the text file record having this key.
KL+7	1	Octal 0377 - Indicates that this is the end of the lowest level index.

The first data sector in an ISI file is a header record used to locate the file from which the index was built. In this way, it is only necessary to specify the name of the index to DATASHARE.

Offset	Length	Description
0	3	PFN and LRN indicators as per DOS convention. See the System Structure Chapter.
3	11	Name of the data file that goes with this index file.
14	3	PFN, and RIB PDA of this file. This field is used to check that the index file has not been moved.
17	3	PFN, and RIB PDA of the file indexed.
23	3	Buffer address and LRN of the last record

used in the data file.

26	3	Buffer address and LRN of the first free index entry.

29.7 Index File Size

The size of the index file for a particular text file may be calculated by the formula below. Applying this formula yields a figure showing the largest size the file might have. The resulting file will generally be smaller than calculated because trailing blanks on keys are discarded, so storage space for the keys is less than expected based on key length.

Calculating the space used by an index file requires calculating the space used for each level of index and adding them. In the following equations:

R = number of logical records to be indexed
L = key length (number of characters per key)
S(i) = number of disk sectors for the i^{th} level of the index tree
K = number of keys per disk sector

For the lowest level index (i = 1),

K = 250/(L+7) (discard any remainder)

and

S(1) = R/K (round up result)

When S(i) = 1 the highest level of index has been reached and the calculation is complete. If S(1) = 1 the calculation is finished in only one step, but only very short files produce a one-sector index. For each successive level of index the following calculation must be performed:

K = 250/(L+3) (discard any remainder)

and

S(i) = S(i-1)/K (round up result)

If S(i)>1, then i=i+1 and perform the calculation again until reachin the highest level of index. Once S(i)=1 has been reached

simply add the space calculated as needed for each level of index:

Total size = S(1) + S(2) + ... + S(i)

Remember that the total space used for the index file will probably be less than calculated here since trailing blanks are not stored.

29.8 Examples of the Use of INDEX

First, a simple example in which only a single ISI file is created, with the same name and on the same device as the text file it indexes. The file is a list of bad checks presented at a local grocery chain, and now each store has a DATASHARE terminal to inquire on the current status of each deadbeat. Thus, while the file is accessed often, additions and deletions are fairly infrequent, so the file will not be reformatted. The file is keyed by bank number (8 digits) and account number (7 digits) concatenated and in positions 1 to 15 of each record.

In order to create the index file, the operator must type:

```
INDEX DEADBEAT;1-15
```

The INDEX program will then create a file DEADBEAT/ISI which DATASHARE can use to access the DEADBEAT/TXT file.

Now, this same grocery chain has expanded its operations, so it desires to include more information on the location and date of each bad check presented. Therefore, they have expanded the file to include the old key in positions 1 to 15, a store location number in positions 16 to 18, and a date field in positions 19 to 24. As an afterthought, the manager decides to tack on the name of the person passing the bad check in positions 193 to 216.

In order to create the indices required for access by any of these keys, the operator must type:

```
INDEX DEADBEAT,BANK;1-15
INDEX DEADBEAT,DATE;19-24
INDEX DEADBEAT,STORE;16-18
INDEX DEADBEAT,NAME;193-216
```

The INDEX program will create four files with names BANK/ISI, DATE/ISI, STORE/ISI, and NAME/ISI. Each file is logically separate, yet all are on the same volume as DEADBEAT/TXT.

Now the store owners have uncovered a hitch - first, the number of bad checks is becoming so large, there is no room on one disk for all the index files and the text file. In addition, access has been slowing way down as the frequency of additions and deletions increases. The store owners have called Datapoint to complain, and their local systems engineer has told them they need to reformat the files when they re-index, and has sold them another disk drive.

The operator now types:

```
INDEX DEADBEAT,BANK/ISI:DR1;FR1-15
INDEX DEADBEAT,DATE/ISI:DR1;19-24
INDEX DEADBEAT,STORE/ISI:DR1;16-18
INDEX DEADBEAT,NAME/ISI:DR1;193-216
```

Note that the reformatting is done only once at the beginning. If reformatting had not been done when the first index was built, it could not be correctly done later without invalidating the previously built indices.

Now, several years later, the grocery chain has expanded and has a large disk system at their main store. The owners are doing so much processing that there is not the time to run the above INDEX programs as each one invokes SORT. However, they wish to keep access time to the minimum. Also, the DEADBEAT file is so large that numerous additions and deletions hardly affect the size.

Every night the operator now types:

```
INDEX BANK;X
INDEX DATE;X
INDEX STORE;X
INDEX NAME; X
```

which recreates the index files. Then during weekly processing, the operator does the processing above which invokes REFORMAT.

The store owners have wisely dispersed some of their data processing to their branch stores. So each night the operator also types:

```
INDEX BANK;K
INDEX DATE;K
INDEX STORE;K
INDEX NAME; K
```

which creates tag files of the four indices. The operator then transmits DEADBEAT/TXT, BANK/TAG, DATA/TAG, STORE/TAG, and NAME/TAG to each of the branch stores. The operator at the branch store, after receiving these files, types:

```
INDEX DEADBEAT,BANK,BANK;I
INDEX DEADBEAT,DATE,DATE;I
INDEX DEADBEAT,STORE,STORE;I
INDEX DEADBEAT,NAME,NAME;I
```

which creates a local set of indices without invoking SORT.

Note: In the above example that created a BANK tag file, the command line with default fields is:

```
INDEX BANK/TXT,BANK/ISI,BANK/TAG;K
```

As only the /ISI and /TAG files are needed for creation of the tag file, the same results could have been achieved by typing:

```
INDEX ,BANK,BANK;K
```

CHAPTER 30. THE INITDISK COMMAND

30.1 Purpose

When a new disk pack is received, it is not immediately usable in the 9370 or 9390 series drives until it has been formatted. The formatting process which causes track and sector identifying information to be written over the entire disk surface, is performed by the INITDISK command. This command is useful only on 9370 or 9390 series disks!

30.2 Use

The INITDISK program is distributed as both a cassette load-and-go (LGO) version and as a disk command program. To invoke INITDISK from a working DOS (normally useful only with two-drive systems), the operator enters at the system console:

```
INITDISK
```

With an LGO cassette, the operator places the cassette in the rear cassette deck and presses the RESTART and RUN keys simultaneously (only the RUN key on a 2200 processor). Once the program has initially loaded, it functions the same regardless of whether it has been loaded from cassette or disk.

After being loaded, INITDISK asks which physical (not logical) drive contains the disk to be formatted and asks the user for confirmation that it is all right to destroy the previous contents of the disk, if any. After the command is satisfied that the user knows what is about to happen, it proceeds to format the disk. The process takes a few minutes.

30.3 Error messages

If the INITDISK command encounters any sort of error indication before or during the formatting process, it will wait for a while to see if the problem will go away on its own. (A typical example would be if the disk to be formatted has not yet come on line when the INITDISK command begins execution). If the problem persists, the program will display a comment on the CRT

display indicating that it is waiting on the disk, describe the status of the disk as indicated by the controller, attempt some corrective actions that may help to clear the situation, and inform the operator of what corrective action has been taken. This is repeated until the problem is successfully cleared up.

CHAPTER 31. KILL COMMAND

31.1 Purpose

The KILL command deletes a file from a logical volume. The file's directory entry is removed and its disk space is deallocated.

31.2 Use

The KILL command is initiated by the operator entering the command line

KILL [<file spec>]

If this file is protected in any way, the message

NO!

will be displayed. If the file specification is not given on the command line (file names which contain special characters cannot be given on the command line), the request for the file name:

```
WHAT FILE?  EXAMPLE:   SCRATCH /TXT:DR1  #143     :DR1
        /   :DR
```

will appear. The user must keyin an eight character filename (including trailing spaces), a slash, a three character extension (including trailing spaces), a colon, the letter "D" and the drive number on which the file resides. If the entire filename specification is not entered properly, the message:

NO SUCH NAME.

will appear. A file can be specified by physical file number by entering "#", followed by the octal PFN, followed by 8 spaces and the drive specification. If the specified file cannot be found (both a name and an extension must always be supplied unless using PFN), the message:

NO SUCH NAME.

will be displayed. If the file exists but it is in a valid subdirectory other than the current subdirectory (except SYSTEM), the message:

THAT FILE IS NOT IN YOUR SUBDIRECTORY

will appear. If the file is found and is not protected, the message:

THAT FILE IS <filename> ON DRIVE n SUBDIRECTORY NAME (xxxxxxxx)

will appear. Then the operator must additionally answer the message:

ARE YOU SURE?

with a 'Y' before the actual deletion of the file is achieved. After the deletion has occurred the following message is displayed:

* FILE DELETED *

CHAPTER 32. LIST COMMAND

32.1 Purpose

The LIST command will list any DOS standard format text file to the screen, a local or servo printer, or a disk file. The command can be used for:

A quick scan of a file by displaying it on the screen (LISTing a file is faster than EDITing it);

Producing a hardcopy listing of a file for permanent records;

Listing a file for use in preparation of a BLOKEDIT command file.

In this chapter, the following terms apply:

Text file means a file with records containing only ASCII characters, except for space-compression bytes and the End-Of-Record and End-Of-File marks. Files created by EDIT and those produced by DATASHARE are normally in the class of text files.

Line means one record of a text file. When displayed on the screen, only the first 72 characters of a record will be displayed; when listed on a local or servo printer only the first 124 characters will be printed. (The remaining eight characters contain a line number.)

Record means the user logical record number (LRN). The first LRN of a file is zero.

32.2 Use

When the LIST program is to be executed, the operator must type:

LIST <filespec>[,<spec2>][,<filespec2>][;options]

Available options are:

L - list on Local printer
S - list on Servo printer
D - Display on CRT
X - suppress line numbers
F - list Formatted print file
P - output formatted Print file
Q - Queue formatted print file, appending to an existing file
Nn - set Number of lines per page to n
I - list in Indexed sequence

Options may be entered in any sequence and should be separated by commas.

32.3 Input File Specification

The file specification (<filespec>) must refer to a DOS text file. If no extension is supplied with the file name, an extension is assumed depending on the options given. A default extension of TXT is assumed unless the option "I" or "F" is used. The option "I" (list a file using its index) causes a default extension of ISI and the option "F" (list a file with format control bytes) causes a default extension of PRT. If no drive is supplied with the file specification, all drives will be searched for the filename/ext. If <filespec> is omitted, the message

NAME REQUIRED.

is displayed. If the file indicated by <filespec> is not found on an online volume, the message

NO SUCH NAME.

is displayed.

32.4 Starting Point

The operator may specify a line number, or logical record number, in the file at which the list should begin by including an optional second parameter <spec2>. For example:

LIST <filespec>,L400

would list the specified file beginning with line 400 of the file.

If the line number specification exceeds the number of lines in the file, LIST returns to DOS after displaying the message:

FILE EXHAUSTED BEFORE LINE FOUND.

LIST <filespec>,R18

would directly access logical record 18 of the specified file and list, starting at line number 1. If range or format errors occur, the error type is indicated and another record number is requested.

For instance, if the record number specification exceeds the number of records, the message

RANGE - NEXT RECORD NUMBER:

is displayed.

The default value for the second parameter is line 1 and record 0.

32.5 Output File Specification

If the options "P" (write to a print file on disk) or "Q" ('QUEUED' write to a disk print file starting at the end-of-file mark) are used, then the third parameter (filespec2) may be used to specify the output file. If the filename is not given, it is assumed to be the same as the input file name. If the extension is not given, it is assumed to be PRT.

Output from either the "P" or "Q" option is a text file with print control characters as described in the Format Control section. The file will be paged with headings; line numbers will be included unless suppressed by the "X" option.

32.6 Output Device

The operator may specify an output device other than the CRT display by including an optional parameter of "S" (servo printer), "L" (local printer), "P", or "Q". For example:

LIST <filespec>,L400;S

would list the specified file on the Datapoint servo printer starting at line 400 or

LIST <filespec>;L

would list the specified file on a Datapoint local printer beginning at line number one.

For either print or disk output LIST will request a heading, which will be placed at the top of every page of output.

The default output device is the CRT display which may be specified by entering a "D".

32.7 Output Format

A parameter is available to suppress line numbers. If the 'X' is entered, lines of up to 132 characters will be printed. For example:

LIST <filespec>;SX

would put the output on the servo printer without line numbers,

LIST <filespec>;X

would display the listing, showing 80 characters per line on the CRT.

Any paged output (from the "L","S","P", or "Q" options) is normally listed at 54 lines per page. The "Nn" option can be used to change the number of lines per page, n being the desired lines/page count.

32.8 Format Control

The parameter "F" is available to allow the handling of print files (those with a format character in the first column of each line). If "F" is entered, the file will be listed without line numbers, page numbers, or headings, since all these items should already be in the print file. The following format characters cause the indicated action to be taken before the line is printed.

1 - Skip to top of form

\+ - Suppress line feed

(space) - Single line feed

0 - Double line feed

- - Triple line feed

Any other character in the first column will be handled as a space (single line feed) and discarded.

32.9 Operator Controls

The listing consists of a continuous stream of the listed file's text. To cause the listing to pause, the operator may hold down the DISPLAY key. To abort the listing, the operator may depress the KEYBOARD key.

32.10 Error Conditions

If printer output was specified and the requested printer is not available, LIST beeps and displays the message:

PRINTER NOT READY

If the printer is made ready, listing will proceed. The KEYBOARD key may be depressed to abort the LIST at this point if necessary.

LIST checks to be sure the text end-of-file is exactly six zeroes and a three (see Text File Formats in the REFORMAT chapter).

If the EOF is not exactly correct, LIST displays the message:

INVALID END OF FILE.

LIST can be used to test for a bad EOF since most text-handling programs are not so particular about EOF format.

When <spec2> has been entered to start LIST at a particular record number, LIST traps FORMAT or RANGE errors and allows a new starting location to be entered. In any other usage, LIST does not trap FORMAT or RANGE errors and any such errors are fatal.

CHAPTER 33. MANUAL COMMAND

33.1 Purpose

The MANUAL command clears the automatic execution established by the AUTO command.

33.2 Use

The command is invoked simply by entering:

MANUAL

If the auto-execution name has not been set the message

AUTO NOT SET.

will be displayed. Otherwise, the System Table location reserved for the auto-execution information will be cleared and the message

AUTO CLEARED.

will be displayed.

CHAPTER 34. MIN COMMAND

34.1 Purpose

The Multiple In (MIN) command is useful for reading multiple files (source, object, Databus object, and relocatable code) from the front cassette drive to disk. It will handle all standard single file (OUT and SOUT), double file (SOBO), and multiple file (LGO, CTOS, and MOUT with or without a directory) tape formats.

34.2 Use

34.2.1 Command Line

The prototype command line for MIN is

MIN [<file1>[,<file2>...]][;<options>]

File specifications are of the form <filename> /<ext>:<drv>. If the drive is not given, all drives online will be searched starting at drive zero. The default extension will be 'TXT' for source, 'ABS' for object, 'DBC' for Databus object files, and 'REL' for Relocatable Code files depending on the tape file format.

34.2.2 Options

Tape file modifications may prevent MIN from automatically determining the tape format. In this event, the options 'L' (for LGO), 'C' (for CTOS), or 'D' (for Directory) are available. Also, option 'N' (for No directory) will tell the system that it is handling a MOUT tape without a directory, which allows entering the file names manually if the directory entry names are not desired. This option also allows entering the directory to disk.

These options are merely test overrides. If, for instance, a tape starts with a recognizable file mark, a loader won't even be tested for and therefore entering the 'L' option is meaningless.

Unfortunately, MIN cannot differentiate an OUT, SOUT, or SOBO tape from a MOUT without directory tape. To speed the processing, the options 'S' (for SOUT) and 'B' (for SOBO) are available. Once again, if the tape doesn't resemble a SOUT tape, for instance, entering an 'S' is meaningless.

MIN accepts a drive specification option ":DRn", ":Dn" or ":<volid>" to force the disk files to a specific drive. Note that this drive specification is an option appearing in the option list following the semicolon, not part of any file specification. Drive specification may be necessary to avoid overwriting existing files on other drives or to force MIN to place the files on a drive other than drive 0.

MIN accepts the 'r' option to direct it to use the rear cassette deck. It will default to the front deck.

If the tape is a MOUT tape with a directory, the options 'A' (for All), 'O' (for Overwrite), 'Q' (for modifying the extension with Q's) are available. Using the option 'A' will load all files on the tape overriding the normal operator message for each file. However, if the file already exists, the operator will be asked if overwriting is desired and if not, for a new file name. Entering the 'O' option in conjunction with the 'A' will force overwriting of existing files (unless write protected). If while processing in the 'All Overwrite' mode a write protected file is encountered, the message:

WRITE PROTECTED

will appear and processing will continue with the next file. Entering the 'Q' option in conjunction with the 'A' will put as many Q's into the directory extension as necessary to create a new filename/ext if the original one already exists. If the original filename/ext exists, the message:

EXISTING FILE

will appear to the right before the modification to the extension is performed. If the filename/QQQ already exists, the message:

Q OPTION EXHAUSTED

will appear to the right and the file will be skipped.

The option 'N' followed by an octal number allows that specific file to be loaded. For example, entering:

MIN FILE/TXT;N12

will load the tape file following file mark 12 (octal) to disk as 'FILE/TXT'. If a non-octal number is entered (e.g. N8) the message:

NUMBER NOT OCTAL

will appear and MIN will be terminated. MIN bypasses the loader on a LGO tape before searching for files. If the file specified is not found, the message:

FILE NOT FOUND

will appear and MIN will be terminated. If the file is found and the file name was not entered on the command line, the file name will be requested as described below.

The options 'L', 'C', 'N', 'S', and 'B' are mutually exclusive. Only one may be entered. The 'A' may be entered with or without the 'D' and with none of the other above options. 'O' and 'Q' are mutually exclusive and may only be entered in conjunction with the 'A'. If any of these restrictions is violated or a character other than those above entered, the message:

BAD OPTION PARAMETER

will appear and the program will be aborted.

34.2.3 Multi-File Named Tapes

34.2.3.1 MOUT With Directory Tapes

Unless overridden with options, these tapes are processed in the follwing manner. The tape is first identified as:

MOUT TAPE FORMAT

Next the date of creation will be displayed:

DATE: DD MMM YY

Then the directory will be displayed:

DIRECTORY: <file1/ext> <file2/ext> <file3/ext>. . .

Then the operator will be asked:

LOAD <file1/ext> ?

If the file is to be loaded the response Y (yes) will cause the file to be loaded. If the response is N (no), the tape will advance to the next file (if any) and repeat the question. If the response is *, control is returned to DOS. If the filename specified already exists, the message:

NAME IN USE. WRITE OVER?

will appear. The answer N (no) will cause the message

DOS FILE NAME:

to be displayed on the same line. The operator must then enter a valid DOS file specification of the form, <name> /<ext>. If the new name already exists the program will loop back to the 'NAME IN USE' message again. When there is no conflict the file will be loaded to disk. The answer > will cause the file to be skipped. The answer Y (yes) will cause the disk resident file to be overwritten. If the file to be overwrittten is write protected, the message:

WRITE PROTECTED OVERWRITE?

will appear. If the response is not Y, the filename request will be displayed again. If the response is Y, the protection is changed from write protect to delete protect and the disk resident file is overwritten. When a file has been loaded from the cassette the message:

LOADED

will appear to the right of the filename. The message:

MULTIPLE IN COMPLETED

indicates the successful completion of the program.

For information on the creation of MOUT with directory tapes, consult the chapter on MOUT.

34.2.3.2 CTOS Tapes

A CTOS tape will be identified as:

CTOS SYSTEM TAPE FORMAT

The system then searches for the catalog (tape file #1). The CTOS file is fairly long so it takes a while. If the catalog file is not an object file or is an object file that loads into memory somewhere other than 017406 or 017410, the message:

BAD CATALOG

will appear and the remainder of the tape will be processed as a multiple numbered-file tape starting at tape file #2. If a good catalog is found, it will then be displayed as:

CATALOG: <file 1> <file 2> <file 3> <file 4>. . .

Then the operator will be asked:

DO YOU WANT TO LOAD <file 1> ?

If the file is to be loaded, the response Y (yes) will cause the message:

DOS FILE NAME:

to be displayed on the same line. If the response is N (no), the operator will be asked for the next file (if any). If the response is *, control is returned to DOS. If no name is entered, the message:

NAME REQUIRED

will appear. If the filename specified already exists, the message:

NAME IN USE. WRITE OVER?

will appear. The answer N (no) will cause the filename request to be displayed again. The answer > will cause the file to be skipped. The answer Y (yes) will cause the disk resident file to be overwritten. If the file to be overwrittten is write protected, the message:

WRITE PROTECTED OVERWRITE?

will appear. If the response is not Y, the filename request will be displayed again. If the response is Y, the protection is changed from write protect to delete protect and the disk resident file is overwritten. When a file has been loaded from the cassette the message:

LOADED

will appear to the right of the filename. The message:

MULTIPLE IN COMPLETED

indicates the successful completion of the program.

34.2.4 Multiple Numbered-File Tapes

LGO tapes and MOUT tapes without a directory are both handled in the same manner. MIN is first executed as:

MIN

An LGO tape will then be identified as:

LGO TAPE FORMAT

In the case of multiple files, MIN will operate in the same manner as described in the section on CTOS tapes above for loading a file without entering the name on the command line. The files, however, will be referenced by number instead of by name. The questions described will be asked for each file on the tape until end of file has been encountered on the tape or an * is entered in response to the "load" question. MIN bypasses the loader on a LGO tape before searching for files.

34.2.5 Double File Tapes

The file specifications for a SOBO tape may be entered on the command line in the following manner:

MIN [<file spec>][,<file spec>];B

File specifications are of the form discussed above. If the second file name is not given, the first name with the assumed extension of ABS will be used. If the extension is not given with the first name, TXT will be assumed. If the filename has not been entered on the command line, MIN will operate in the same manner

as described in the section on multiple numbered-file tapes above for each file on the cassette, displaying the messages in the same order for both files.

34.2.6 Single File Tapes

For OUT, and SOUT tape formats, the file specifications may be included on the command line in the following manner:

MIN [<file spec>];<option>

where <option> is an 'S' for SOUT tape formats.

If the file name has not been entered on the command line, the program will ask for the file name in the manner described in the section on multiple numbered-file tapes.

34.3 Tape Formats

MIN will automatically process the tape format by the following conventions if an option is given.

34.3.1 Single File Tapes

An OUT (object out) tape format has a file mark zero, a file mark one, an object file with entry point, and a file mark 0177. An object file has an address with the MSB and LSB in the fourth and fifth bytes of each record. Their complements are in the sixth and seventh bytes. The remainder of each record is filled with octal characters (ranging from 0 to 0377).

A SOUT (source out) tape format has a file mark zero, a source file, a file mark one, and a file mark 0177. A source file consists of records containing only ASCII characters, except for space compression bytes, physical end-of-record bytes, and logical end-of-record bytes.

34.3.2 Double File Tapes

A SOBO (source and object out) tape is the combination of a SOUT and OUT tape. It has a file mark zero, a source file, a file mark one, an object file with entry point, and a file mark 0177.

34.3.3 Multiple Numbered-File Tapes

A LGO (load and go) tape has a loader, a file mark zero, a string of files (the first being an object file and the rest may be source, object, Databus object, and relocatable code intermixed) separated by sequential file marks, and a file mark 040.

A MOUT (multiple out) tape without directory has a file mark zero, a string of files (may be source, object, Databus object and relocatable code intermixed) separated by sequential file marks, and file marks 040 and 0177. Single and double file tapes are included in this category if options are not used.

34.3.4 Multiple Named-File Tapes

A CTOS (cassette tape operating system) tape has a loader, a file mark zero, a CTOS object file with entry point, a file mark one, a catalog object file, a string of files separated by sequential (though not necessarily contiguous) file marks, and a file mark 040.

A MOUT (multiple out) tape with directory has a file mark zero, a tape directory, a string of files separated by sequential file marks, and file marks 040 and 0177. The directory is a source format file containing a date entry seven bytes long (DDMMMYY) and 31 file name entries each eleven bytes long (eight bytes for the name and three bytes for the extension). The entries are separated by end-of-string bytes (octal 015). This makes it convenient for display under CTOS LIST or to load to disk and list.

34.4 Errors

If the tape format is not one of the eight standard formats outlined above in the Tape Formats section (e.g. it starts with a file mark two) the message:

INVALID TAPE FORMAT

will appear and the processing will be aborted.

If the end of tape is detected while processing, the message:

END OF TAPE

will appear and the processing will be aborted.

If a parity error is encountered in an object or Datashare file on tape, the message:

PARITY ERROR-FILE WILL BE DELETED

will appear, the file name will be removed from the disk directory, and processing will skip to the next file. If a parity error is encountered in a source file on tape, the message:

PARITY ERROR-RECORD MODIFIED

will appear, a 253 byte disk record will be written with percent signs in the first five positions of the record data, and processing will be continued with the next record.

If an unrecognizable record format is encountered, the message:

UNRECOGNIZABLE TAPE RECORD FORMAT

will appear and MIN will be terminated.

CHAPTER 35. MOUT COMMAND

35.1 Purpose

The Multiple Out (MOUT) command is useful for writing multiple (up to 32, or 31 if a directory is used) disk files (source, object, and Datashare) out to the front cassette drive.

An additional feature is the ablity to create a tape file directory as file #0 on the tape. The directory is a source format file, that is, it consists entirely of ASCII characters except for space compression bytes, physical end-of-record marks, and logical end-of-record marks. The directory contains a date entry seven bytes long (DDMMMYY) and 31 file name entries each eleven bytes long (eight bytes for the name and three bytes for the extension). The entries are separated by end-of-string bytes (octal 015). This makes it convenient to list under CTOS LIST or to load to disk and list. The directory is also used by the MIN program to enter files to disk. MOUT creates the directory in memory before the tape writing starts even if it is not to be written to tape. The writing of a full tape (over 500 records) takes about 10 minutes, which shows the advantage of entering all the names before writing begins.

Another feature is the option to automatically verify a tape following its creation. Or a previously written directory tape may be verified in an 'only verify' mode. If this mode is requested, the system will read the directory on the cassette tape in the front drive (if a valid directory is not found, the system will request file names from the operator) and verification will be performed against the indicated files.

35.2 Use

File specifications and/or options may be entered on the command line in the following manner:

MOUT [<file spec>,<file spec>,...][;options]

File specifications are of the form FILENAME/EXT:DR#. If the drive is not given, all online drives will be searched starting at drive zero. If the extension is not given, ABS is assumed. File

specs are separated by anything (including multiple spaces) except letters, numbers, slash (/), or colon (:).

Options (which follow a semi-colon and may be spaced or separated by commas) are 'L' for a loader format tape, 'D' for a directory format tape, 'V' for verification of the created tape, 'X' for verification only, and 'r' (lower-case letter actually entered) for output to rear cassette.

If a loader is to be written, the first file (file #0) must be an object file. There are no restrictions on files other than #0.

The directory option ('D') will write a tape directory as file #0. The first item within the directory is the date entered DDMMMYY. The month is entered as three alpha characters. The date may be entered following the option letter (for example, D12JAN74). If the date is not entered, it will be requested.

The verify option ('V') will verify all the files on the created tape. Verification consists of making a byte for byte comparison between the data on the disk and the data on the tape. If verification fails, the tape will be rewritten and verification tried one more time.

The verify only option ('X') will cause the first tape file to be read from the front deck. If file #0 is a directory (first seven characters of DDMMMYY format), the remaining files will be automatically verified using the directory entries. If the tape begins with a loader, it will be verified and file names requested for the remaining files as they are verified. An 'N' may be entered immediately preceding the 'X' to force the system not to recognize the directory. This would be done if manually entering file names is desired (for instance, the directory names don't match the disk file names). If there is neither a directory or loader, file names are requested as the files are verified.

If the semi-colon is entered with no entry following, it will be interpreted that the tape will not have a loader, a directory, or any verification.

Entering 'D' and 'L' together or entering anything with 'X' other than 'N', or entering some letter other than 'D', 'L', 'V', 'X', or 'N' will result in the message:

BAD OPTION PARAMETER. MOUT DISCONTINUED.

and the Multiple Out will be aborted.

If file names and/or options are not entered on the command line, MOUT will ask for them as required. If options were not entered, the first question will be:

DO YOU WANT A LOADER?

Replies other than 'Y' or 'N' will be answered by:

WHAT?

and a repeat of the question. If the reply is 'N', the next question is:

DO YOU WANT A DIRECTORY?

Again, if the reply is other than 'Y' or 'N', it will be answered by:

WHAT?

and a repeat of the question. If the reply is 'Y', the next request is:

ENTER THE DATE (DDMMMYY):

where the month is entered as three alpha characters. If the day is not in the range of 00 to 39, the month not alpha, or the year not in the range of 70 to 99, the response:

BAD DATE

will appear and again the request for the date. The next question is:

DO YOU WANT TO VERIFY THE TAPE?

If the reply is not 'Y' or 'N', the response:

WHAT?

will appear followed by a repeat of the question. If the reply is 'Y' and the replies to the loader and directory questions were 'N', the question:

DO YOU WANT TO ONLY VERIFY THE TAPE?

will then be asked. If the reply is other than 'Y' or 'N', the response

WHAT?

will appear followed by a repeat of the question. If only verification is requested, the first tape record on the front tape deck is read in. If it is a directory (the first seven characters of DDMMMYY format), the remaining tape files will be automatically verified using the directory entries. If it is a loader, the message:

LGO TAPE FORMAT

will appear. The message:

LOADER IS BEING VERIFIED

will then appear as the loader is being verified. If the loader verifies correctly, the message:

LOADER OK

will appear to the right. Otherwise, the message:

BAD LOADER

will appear. After checking the loader, or if the tape has neither a loader or directory, the message:

CASSETTE FILE #XX (format) DOS FILE NAME:

will appear where XX is the file number and (format) is (SOURCE), (OBJECT), (DATABUS CODE), or (RELOCATABLE CODE) depending on the file format. If nothing is entered, the message:

NAME REQUIRED

will appear and the request will be repeated. If an asterisk (*) is entered, MOUT will terminate and return to DOS. If a greater-than sign (>) is entered, the program will skip to the next file. If a less-than sign (<) is entered, the program will backspace to the prior file (bypassing null files). If the program finds the beginning of the tape, it will beep and then move forward to the first file. If a name is entered, the default extension is 'TXT' for source, 'ABS' for object, and 'DBC' for Datashare object depending on the file format. If the drive number is not entered, all online drives will be searched starting at drive zero. If a drive number greater than DOS allows is given, the message:

BAD DRIVE

will appear and the request repeated. If the file is not found, the message:

FILE NOT FOUND

will appear and the request repeated. If the disk file is found, it will be matched byte by byte against the disk file. If the files completely match, the message:

FILE OK

will appear to the right and processing continues with the next file. If an error is detected, the appropriate message will appear and processing continues with the next file. Null files are bypassed. Processing continues until an end-of-tape mark (file mark 040 or 0177) is read at which time the message:

VERIFICATION PHASE COMPLETED

will appear and MOUT will be terminated.

Use of the 'r' option does not change the program operation described above, it simply causes the rear cassette deck to be used rather than the front deck.

35.3 File Names

If the file names are not given in the command line, the operator will be asked for the file names one at a time. The request is of the form:

CASSETTE FILE XX DOS NAME:

where XX is the file number. Possible replies to the file name query include:

a) the file specifications as discussed above,
b) a pound sign (#) which will bump the file number to 20 octal if not already there (only allowed on loader tapes to initiate numbered files on a CTOS tape),
c) a dollar sign ($) which will cause a null file (tape file mark only) to be written to tape and the file spec of NULL/NUL to be entered in the directory,
d) an asterisk (*) which will indicate no more files are to be

entered and the tape writing started (writing is postponed until the directory is complete), and

e) OS which will abort the program. The message: MULTIPLE OUT DISCONTINUED will appear and control is returned to DOS. (To dump OS/ABS, enter 'OS/ABS').

If the operator fails to enter a name, the message:

NAME REQUIRED

will appear and the name request will be repeated. If the drive is given and is not in the range valid for DOS, the message:

BAD DRIVE

will appear followed by a re-request of the name. If the file is not found, the message:

FILE NOT FOUND

will appear followed by a re-request of the name. If the file is found, the format (object, source, or Datashare) will be determined by the system. If the tape is a loader tape and file #0 is not an object file, the message:

FILE FOLLOWING LOADER NOT OBJECT

will appear along with a re-request of the file name. This message may also be displayed if the reply to the file name query for file #0 is a pound sign. Otherwise the messages:

OBJECT FILE

or:

SOURCE FILE

or:

DATABUS CODE FILE

or:

RELOCATABLE CODE FILE

or:

NULL FILE

will appear to the right of the file name. If the pound sign is entered for a tape that does not have a loader, the message:

NOT LGO TAPE

will appear with a re-request of the file name. If 32 files (or 31 on a directory tape) are entered, the message:

THAT'S THE END OF THE LINE

will appear and the tape writing is started automatically.

35.4 Writing

Once the tape writing has started, the system will keep the operator informed of its progress. As a loader is being written, the message:

LOADER IS BEING WRITTEN

will appear. As a directory is being written, the message:

DIRECTORY IS BEING WRITTEN

will appear. While files (including null files) are being written, the message:

FILE <filename/ext> IS BEING WRITTEN

will appear. When the writing is completed, the message:

WRITING PHASE COMPLETED

will appear.

If a non-object record is sensed in an object file while writing to tape, the message:

FILE CONTAINS NON-OBJECT RECORD

will appear and the next file is written over the bad tape file including the file mark. This will leave a directory entry without a file. If this should happen, it will cause verification to display the message:

NON-SEQUENTIAL FILE MARK

and the tape rewritten.

If a non-source record is sensed in a source file while writing to tape, the message:

INCORRECTLY FORMATTED SOURCE RECORD

will appear. The file is ended at this point without writing the bad record and the next tape file will start immediately following. If this should happen, it will cause verification to display the message:

INCORRECTLY FORMATTED DISK RECORD

or:

TAPE EOF BEFORE DISK EOF

and the tape rewritten.

If MOUT runs out of tape, the message:

END OF TAPE ENCOUNTERED WHILE WRITING filename/ext

will appear, an end of tape marker written at the end of the previous tape file, and the unwritten files will be removed from the directory (if there is one). Processing then will be continued with verification.

35.5 Verifying

If verification is requested, the system will keep the operator informed of its progress. As a loader is being verified, the message:

LOADER IS BEING VERIFIED

will appear. As a directory is being verified, the message:

DIRECTORY IS BEING VERIFIED

will appear. While files (including null files) are being verified, the message:

FILE filename/ext IS BEING VERIFIED

will appear. When the verification is completed, the message:

VERIFICATION PHASE COMPLETED

will appear. If verification is requested for a tape having no directory, the message:

NOT DIRECTORY TAPE

is displayed. Then the message:

CASSETTE FILE #XX(format) DOS FILE NAME:

will appear. The filename should be entered. Responses are discussed in the section under OPTIONS.

A variety of error messages may be displayed during the verification phase. Most of them are self-explanatory. They include:

BAD LOADER

BAD DIRECTORY

TAPE FILE DOES NOT MATCH DISK FILE

INCORRECTLY FORMATTED DISK RECORD

DISK FILE CONTAINS NON-OBJECT RECORD.

DISK FILE CONTAINS NON-TEXT RECORD.

NON-SEQUENTIAL FILE MARK.

TAPE FILE MARK READ BEFORE TAPE OBJECT EOF.

TAPE OBJECT EOF NOT FOLLOWED BY TAPE FILE MARK.

DISK EOF BEFORE TAPE EOF

TAPE EOF BEFORE DISK EOF

If an error is detected, the program will then either rewrite the tape (if it has just been created) or skip to the next file (if in the 'verify only' mode). If it rewrites the tape, the message:

I'M NOW REWRITING THE TAPE

will appear. The system will rewrite once before quitting completely at which point the message:

VERIFICATION UNSUCCESSFUL

will appear and the processing terminated.

If a problem arises that causes an abnormal end (e.g. end of tape), the message:

MULTIPLE OUT DISCONTINUED

will appear, otherwise the message:

MULTIPLE OUT COMPLETED

will signal the successful end of the program.

ERROR D ON DECK 2

will signal parity errors on the cassette and control is returned to DOS.

CHAPTER 36. NAME COMMAND

36.1 Purpose

The NAME command allows the user to change the name of a file, the extension of a file, or the subdirectory in which a file resides. However, the contents of the file and the volume on which it resides are not affected in any way.

36.2 Use

The NAME command is invoked by the operator entering a command line of the form

NAME <file spec1>[,<file spec2>][,<subdirectory name>]

The first file specification refers to the current file name and the second file specification is the new name and/or extension to be assigned. If no extension is supplied in the first file specification, ABS is assumed. If no extension is supplied in the second file specification, the extension of the first file is assumed. If no filename is given in the second file specification, the name of the first file is assumed. The drive number should only be specified in the first file specification.

If the NAME command is used to move a file from one subdirectory to another the second file specification may be omitted (unless the filename and/or extension are to be changed) and the subdirectory name denoting the subdirectory into which the file is to be placed is the third specification:

NAME <file spec1>,,<subdirectory name>

If no subdirectory name is entered, the file is placed in the current subdirectory.

If the first file specification is not given, the message

NAME REQUIRED.

will be displayed. If the second name is already defined on the drive that contains the first file, the message

NAME IN USE.

will be displayed. Note that the drive specification on the second name is ignored. If the first name is not found on an online disk, the message

NO SUCH NAME.

will be displayed. If the subdirectory name keyed is not found on the disk containing the file to be renamed, the message

NO SUCH SUBDIRECTORY.

will be displayed.

CHAPTER 37. PUTIPL COMMAND

37.1 Purpose

The PUTIPL command writes an IPL (Initial Program Loader) block and DOS boot blocks to the disk.

37.2 Use

PUTIPL <:DRIVE>

If the drive number is not specified in the command line, PUTIPL will display the following:

LOGICAL DRIVE TO BE WRITTEN (0-max OR "*" TO EXIT TO DOS):

Respond with the drive number that you want to write to.

If you are running under ARC, PUTIPL may be executed on local drives. Attempting to write IPL blocks on a remote volume, however, will result in the error:

NOT TO A REMOTE VOLUME

CHAPTER 38. PUTVOLID COMMAND

38.1 Purpose

The PUTVOLID command writes a symbolic volume identification (VOLID) onto a disk.

38.2 Use

PUTVOLID <volid> <:drive>[;<owner id>]

Where <volid> is 1 to 8 characters in length, <drive> is the logical drive to be written to, and <owner id> is any information the user wants.

If only a drive number is entered, the existing <volid> for that drive will be displayed.

If the information on the command line is incorrect in any way, an error message will be displayed with the proper format to enter.

PUTVOLID may be used under ARC to write a VOLID on a local volume. However, attempting to write a VOLID on a remote volume will result in the error:

NOT TO A REMOTE VOLUME

CHAPTER 39. REFORMAT COMMAND

39.1 Purpose

The DOS REFORMAT command is used to change the internal disk format of text-type (non-object) files. Additionally, it can recover disk space left unused when files are updated by the DATASHARE indexed sequential access method. REFORMAT can compress a file in place on disk provided that such compression does not entail the writing of a physical disk sector prior to the time that sector is read. REFORMAT maintains logical consistency in such cases and will not write on a disk file until it has checked to be sure it can complete its job successfully.

39.2 Use

When the REFORMAT program is to be executed, the operator must type:

REFORMAT <file-spec>[,<file-spec>][;<parameters>]

where only the first file specification is mandatory, and specifies the file to be reformatted. If the second file specification is given, it must be distinct from the first. Reformatting in place is requested by omitting the second file specification.

The parameter list describes the format the output file is to take, and whether REFORMAT is to free any disk space that might be vacated by the reformatting process. In addition, the user can specify that REFORMAT is to pad short records, and either truncate or segment long records. REFORMAT will produce three different kinds of output files: record compressed, space and record compressed, or blocked records (see the section on TEXT FILE FORMATS). Note that REFORMAT will not produce blocked space compressed records or space compressed non-record compressed files although such files can be used as input to the REFORMAT program. If no parameters are given, the output file is blocked one record per sector.

Parameters passed to REFORMAT may be separated by spaces or commas. The valid parameters are as follows:

Parameter	Description
B<n>	The output file will be blocked. This implies no space or record compression, with <n> logical records per physical sector.
C	The output file will be space and record compressed. The number of logical records per physical sector will be indeterminate.
R	The output file will be record compressed, but no space compression will be done. In general, the number of logical records per physical sector will be indeterminate.
L<n>	The length of each logical record will be adjusted to <n> (up to 65,000) characters. Note that if the logical records are space compressed, this will not make the physical length of the records <n> characters. If the logical record is shorter than <n> characters, it will be padded with blanks to the proper length. If the logical record is longer than <n> characters, the action taken depends on the T and S parameter. Note that when L is greater than 245, the output must be record compressed.
T	(Only valid if L parameter is given) Truncate the logical record if it is longer than <n> characters.
S	(Only valid if L parameter is given) If the length of the logical record is greater than <n> characters, segment it into (q) logical records each of length <n>, padding if necessary. The number (q) is defined as input length divided by <n> rounded upward to the next integer. If neither S or T is specified, and an input record of length greater than <n> is found, a message is issued and REFORMAT gives up.

D If reformatting is done in place and this parameter is specified, any disk space vacated by the reformatting process will be returned to the operating system for re-use.

39.3 Output File Formats

The REFORMAT utility permits you to select essentially three different output file formats. It will produce blocked files that are not space compressed, record compressed files that are not space compressed, and files that are both record and space compressed. In addition, it has a subcommand to permit you to specify the logical length of the output records. Use of this subcommand will guarantee that each record has exactly the same logical length. Note that if the output format does not specify space compression, the physical length of each record will be identical. This is especially useful for telecommunications disciplines that require records of fixed length.

If you have set a fixed logical length for output records, there are two subcommands available to tell REFORMAT what to do with records whose logical length exceeds the specified output length. You may select either truncation of the input record, or you may segment it into two (or more) output records, each of the logical length specified.

39.4 Reasons for Reformatting

Several uses of REFORMAT deserve special mention. First, a random disk file is structured to have one logical record per physical sector. Often, however, it is convenient to create a random file through the use of the general purpose editor - which record and space compresses its output. REFORMAT can then reprocess the file into the correct format for DATASHARE or DATABUS random access.

Secondly, when a file is accessed with DATASHARE indexed sequential access method, any additions or deletions result in an increase in the physical size of the file. The reason for this is that any inserted records are placed at the physical end of the file, and each one consumes at least one entire physical sector, regardless of its logical length. Similarly, deleted records are simply overstored with octal 032 (logical delete) characters, and the space they vacate is not reused. REFORMAT recognizes this condition, and will recover such vacated space. Note that ISAM read-only or update-only (no additions or deletions) files do not usually need reformatting.

39.5 Reformat Messages

The REFORMAT utility program produces several messages on the operator's console. The contents and where necessary, meaning of those messages follow:

DOS. VER 2 REFORMAT COMMAND - date
Self-explanatory sign on message.

COMMAND LINE ERROR: 015 missing
This is an internal error and should be reported to Datapoint.

PROGRAM ERROR - EXCESS FILE SPACE NOT DEALLOCATED TO PREVENT POSSIBLE LOSS OF DATA
REFORMAT has detected an invalid end of file mark. In order to prevent the possible loss of data which might be after the invalid end of file indicator, space allocated but unused is not freed.

EXCESS FILE SPACE NOT DEALLOCATED; OUTPUT FILE IS DELETE PROTECTED.
Self-explanatory.

OUTPUT FILE IS WRITE PROTECTED AND CANNOT BE WRITTEN INTO OR SHORTENED.
You have requested REFORMAT to output to a write-protected file.

INVALID OPTIONS SPECIFIED
You have given REFORMAT an invalid parameter list. This message is followed by the valid options you may specify.

ILLEGAL, CONFLICTING OR DUPLICATE OPTIONS
   You have specified two mutually exclusive options.

YOU SPECIFIED BOTH SEGMENTATION AND TRUNCATION.
YOU CANNOT HAVE BOTH
   Self-explanatory.

BLOCKING FACTOR CONTAINS INVALID NON-NUMERIC DIGITS
   Self-explanatory.

BLOCKING FACTOR REQUIRED BUT MISSING OR ZERO FOUND
   You specified blocking but omitted the blocking factor.

LOGICAL RECORD LENGTH REQUIRED BUT MISSING OR ZERO FOUND
   You must specify the logical record length of the output file if you wish to have fixed length output records.

YOU HAVE ILLEGALLY ENTERED A SPECIFICATION FOR
A THIRD FILE
   REFORMAT recognizes only two file specifications.

HOW DO YOU EXPECT TO FIT THAT MANY RECORDS IN A
256 BYTE SECTOR?
   Self-explanatory.

LOGICAL RECORD LENGTH MUST BE < 65535 BYTES.
   Self-explanatory.

YOUR BLOCKING FACTOR IS TOO LARGE FOR THE SIZE
OF THE RECORDS YOU HAVE.
   Self-explanatory.

YOUR LOGICAL RECORD LENGTH IS TOO SMALL FOR THE
SIZE OF THE RECORDS YOU HAVE
   While processing the input file, REFORMAT came across a record that was larger than the specified logical record length. Since you specified neither segmentation nor truncation, this is recognized as an error.

SPECIFIED OUTPUT FILE FORMAT ENLARGES PRESENT
INPUT FILE. FILES CANNOT BE ENLARGED DURING
REFORMAT-IN-PLACE. REFORMAT IN-PLACE REQUEST
REFUSED.
   Self-explanatory.

YOU SPECIFIED AN OUTPUT FILE THAT ENDED UP
BEING YOUR INPUT FILE. TO REFORMAT IN-PLACE
DO NOT SPECIFY ANY OUTPUT FILE.
   Self-explanatory.

OUTPUT FILE NOT FOUND ON DRIVE X.
OUTPUT FILE FOUND ON DRIVE Y.
OUTPUT FILE WILL BE CREATED ON DRIVE Z.
   These messages only occur if no specific drive was indicated for the output file. The first message appears followed by either the second or third. REFORMAT could not find the output file on the same drive as the input file. It either found one on a different drive, or created one on the displayed drive. If the output file is created, it is always created on the same drive as the one the input file is on.

REFORMAT IN-PLACE REQUESTED.
PRESCAN IN PROGRESS.
   REFORMAT is checking to make sure it can properly process the file inplace.

FILE ALREADY WAS IN THE SPECIFIED FORMAT
   Self-explanatory.

COPYING WITH REFORMATTING IN PROGRESS
   Self-explanatory.

REFORMAT-IN-PLACE IS IN PROGRESS.
DO NOT DISTURB!!!
   Self-explanatory.

NAME REQUIRED
   Either you gave only an extension or drive for the input file, or you specified the output file first, followed by the input file.

INVALID DEVICE
   An invalid drive was specified for the input file.

NO SUCH NAME
   The input file specified cannot be found.

INVALID DRIVE SPECIFICATION
The drive specification entered for one of the file specifications was not in a valid format.

39.6 Text File Formats

Under Datapoint Corporation's Disk Operating System, text files consist of legal ASCII characters, which make up the text itself, and various control characters with special meanings. It is illegal to have the control characters in the text portion of the file. According to DOS convention, any character between 000 and 037 is considered a control character.

Each physical record of a text file is a logical disk sector, and contains 256 characters. The first three and last two characters are reserved for control functions; hence, the maximum space available in a single physical record is 251 bytes. The format of a logical sector is as follows:

Offset (octal)	Length (octal)	Description
000	001	Physical file number of this file. For a detailed description of physical file organization, see the chapter on System Structure.
001	002	Logical record number. This refers to logical physical records, and is not related to text records within the file.
003	373	Text. 251 bytes of text and control characters, depending upon the format of the file.
376	002	Two characters reserved.

The text part of each file is considered a logical stream, crossing sector boundaries without being logically discontinuous. Demarcations of logical record boundaries are made solely by control characters imbedded within the text itself. There are essentially five control characters found in files generated by DOS: 000 <NUL> used for end of file indication, 003 <EM> used to denote the end of medium (a sector boundary) but not the end of a logical record, 011 <CMP> used to denote space compression, 015 <ENT> used to denote the end of a logical record, and 032 <DEL> used to denote deleted data.

Under DOS each file is treated as a single, continuous stream of data. Physical records bear no relation to the logical structure of the data contained in them. In this way, a proliferation of different file structures and the special routines needed to treat such special cases have been avoided. This does not mean that there cannot be a relation between physical and logical structure, it simply means that such a relationship is incidental to a particular file, and need not be treated as a special case. For example, random access to a data file is defined in the DATABUS language. Files to be accessed in this manner are structured in such a way that one logical record corresponds exactly with one physical record. This structure is not inherent in the makeup of a random file, in fact, such files can be treated exactly as any other text file.

The basis for this treatment of text files is the logical record. A logical record starts at the beginning of a file, or immediately after the end of a previous logical record. It consists of ASCII data and is of no pre-determined length. Instead, the record is terminated with a single ENT character. In this way, complications arising from a multitude of record types are entirely avoided.

If the logical record contains any CMP characters, it is said to be space-compressed. The character immediately following the CMP character is a space count, and the pair represent the number of ASCII blanks removed when the record was compressed. Since the character following CMP is always assumed to be a space count, CMP can never occur as the next-to-last text character in a physical sector, since the EM character following it would be lost.

If the file is organized so that each physical sector contains exactly the same integral number of logical records, with no logical record spanning an EM character, the file is said to be blocked. If the file is not blocked, then it is said to be record compressed. Note that for a blocked file all sectors except possibly the last one in the file contain the same number of logical records while for record compressed files the number of logical records per physical sector is indeterminate.

Under DOS conventions, a valid end of file mark consists of exactly six NUL characters, followed by an EM character:

000 000 000 000 000 000 003

This mark must begin at a sector boundary. All information after a valid end of file mark in the sector is indeterminate.

CHAPTER 40. REWIND COMMAND

40.1 Purpose

The REWIND command causes the front or rear cassette deck to rewind the cassette in place.

40.2 Use

The REWIND command is entered as:

REWIND [<deck>]

If the <deck> entry is not made, the cassette in the front deck will be rewound. If <deck> is specified the cassette in the rear deck will be rewound. The actual values entered for <deck> are "REAR" or "DECK1". If some unrecognizable entry is made for <deck> the entry will be ignored and the front deck will be rewound.

REWIND returns to the operating system as soon as the cassette rewind operation has been started. If no cassette is in place in the specified deck the error message:

NO CASSETTE IN SELECTED DECK.

The cassette can be fully wound onto the clear leader at the reverse end of the tape, since the rewind command starts by slewing the tape backwards for a few seconds first. This action takes up any slack that may be present in the cassette before the high-speed rewind starts, and also ensures that the tape is not on the clear leader when the actual rewind begins.

CHAPTER 41. SAPP COMMAND

41.1 Purpose

The SAPP command allows appending two text files to create a single larger file, or copying one text file to another, recognizing the end of file and not copying unused file space.

41.2 Use

SAPP <file spec>,[<file spec>],<file spec>

The SAPP command appends the second source file after the first and puts the result into the third file. If extensions are not supplied, TXT is assumed. The first two files must exist. If the third file does not already exist, a new file will be created. The first file's end of file record is discarded and the copy is terminated by the end of file mark in the second file.

Omitting the second file specification causes the first file to be copied into the third file. Note that neither the first or second file is changed.

The first and third file specifications are required. If either is omitted the message

NAME REQUIRED

will be displayed.

The second and third file specifications must not be the same.

CHAPTER 42. SORT COMMAND

42.1 Purpose

The Disk Operating System SORT provides a facility allowing any Datapoint disk user to initiate file sorts directly from the keyboard. The SORT can resequence the records of a file in ascending or descending order based on a key defined by a set of character column positions. SORT provides considerable control of the output record format, and can even output directly to a local or servo printer.

The FASTSORT utility, release separately, will resequence a file much more quickly than SORT for users with 5500, 6600 or 1800 series processors. FASTSORT does not provide the full capabilities of SORT, especially the limited output facility.

For more sophisticated uses, SORT may be called from other programs through CHAIN. Using CHAIN also enables complicated sort options to be reduced to a single file name which may then be invoked either from the keyboard or another program. CHAIN also extends the SORT package to operate as a merge.

42.2 Use

SORT is invoked using a command line of the format:

SORT <in>,<out>[,<drv>][,<seq>][;<options>]

<in> is the input file
<out> is the output file
<drv> is the SORT work file drive specification
<seq> is the collating sequence file
<options> include specifications for the following:
    output file format (I, C or N)
    special output record format (L, T, or K)
    hardcopy output (H or X)
    collating sequence order (A or D)
    primary/secondary record specification (gnnntc)
    sort key specifications (sss-eee)

All the fields of the command line are explained in detail

later in this chapter.

42.3 Fundamental SORT Concepts

42.3.1 File Formats

All Datapoint systems use a universal text file structure recognized by Databus, COBOL, RPG, Basic, Scribe, Editor, Assembler, terminal emulators, and so on. Therefore, any text file generated by or for any of the above may be sorted. The file to be sorted must be on disk, however.

There are two sub-formats a Datapoint file can have: Blocked or Sequential. Blocked files are required to have a single 'string' or 'record' of data per physical disk record. The maximum record size for blocked records is 249 bytes (plus end-of-record and end-of-sector control bytes for a total of 251 bytes). Sequential records have no fixed relationship to physical disk records and are written as densely as possible in the given file space. Nonetheless, blocked files can be read sequentially in the identical way that sequential files are read. In fact, both types of files, when read sequentially, are indistinguishable. Blocked files are used for achieving random access to records. They generally require more disk space than sequential files for the same amount of data.

Space compression implies that the logical position and the physical position of a character in a record may differ. SORT will always expand the spaces to determine the logical position of a character.

When sorting, consider that the result of the sort is not a restructuring of the original file. It is a new file which is a restructured copy of the original file. The original file is never changed.

Therefore, SORT produces a file which is a sorted version of the original. This gives the user the added opportunity of specifying the type of file to be output regardless of the input file format (with some restrictions; see the section on Input/Output File Format Options).

42.3.2 The Key Options

The KEY of a sort is the field or that part of the record which is to order the sequence of records. For instance, it can be a person's name, state, employee number, amount in debt or any aspect of the data base identifiable by a fixed position in the record, based upon the column count from the beginning of the record.

Consider the following record (column count scale below for reference only):

```
Mule, Francis A.       242219 123 BARN    SAN ANTONIO     TX
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

The name begins in column 1 and goes to 22. The employee number spans columns 24-29. The street address is 31-42. The city is 43-58. The State is 59-60

If each person had a record in the file exactly in the above format, SORT could order the sequence of records in the file by any of the above fields. For instance, to get an alphabetical list of the records by name, the key would be 1 to 22 (hereafter referred to as 1-22). The key for sequencing the file in order of employee number would be 24-29. The key for ordering the records by state then city and then employee number would be 59-60,43-58,24-29.

Any portion of the record can be used as a key. Care must be taken when selecting a key to include no more characters than necessary, since each character added to the key slows down the sort.

The key specified for SORT is concatenated to a single string, then sorted character-by-character, with the left-most character being of most significance. It is very important to realize the effects of a right-to-left character sort. To appear in the correct sequence, numeric fields must be right-justified, character fields must be left-justified. If signed numeric fields are sorted, the sign should be moved to the left-most position and the magnitude right-justified; otherwise the resulting sorted sequence will contain positive and negative values in no discernible order, since the "-" and "+" signs are just another character to SORT. A full explanation of character sort concepts is beyond the scope of this manual. Interested users should consult an appropriate information science textbook.

42.3.3 How to Sort a File

File sort operations are initiated by a single command line. All the operator must know is the name of the file to be sorted, the name desired for the sorted output file, and the columns containing the key.

For instance, the keyboard issued command for the above example to sort on the name field (1-22), would be:

```
SORT EMPLFILE,SORTFILE;1-22
```

This is assuming that the name of that file was EMPLFILE. It is also the operator's decision as to what the resultant sorted file is called, as the command could have easily been:

```
SORT EMPLFILE,EMPSORT;1-22
```

as well. The second file named is where the resultant sorted output will be placed.

More complicated keys may be stated as well and the command to sort by state and then name would be:

```
SORT EMPLFILE,SORTFILE;59-60,1-22
```

That is all there is to simplified sorting.

Testing SORT for yourself is simple. Most systems have a source code file for a Databus or Assembly language program on the disk. Such programs can be sorted by op-code and provide an interesting analysis of the usage of each instruction type:

```
SORT INFILE,OUTFILE;9-12
```

42.4 SORT Command Line and Options

42.4.1 Generalized Command Statement Format

The following is the generalized statement format for the Datapoint DOS SORT:

SORT <in>,<out>[,<drv>][,<seq>][;[f][o][r][h][gnnntc][k_1]. . .[,o_n][,k_n]

In the options field commas may be used to separate

parameters; commas must be used to separate sort key groups (k_n) and order specifications (o_n) when more than one of either is used. The fields and option parameters definable on the SORT command line are:

<in>. . . . . Input file specification. This file must exist on disk. The default extension for the input file is 'TXT'.

<out> . . . . . Output file specification. If the specified output file already exists, it will be overwritten. The default name of the output file is the same as that entered for the input file. The default extension of the output file is the same as that entered or assumed for the input file. The output file cannot be the same as the input file.

<drv> . . . . . SORT work file drive specification. Only a drive specification of the form :Dn or :<volid> is legal in this field; entering a name or extension will cause SORT to abort. SORT attempts to optimize its speed by placing its work files on a drive separate from the input or output files. Unless otherwise directed by this field, SORT creates its work files on the first available disk, starting with the highest-numbered drive online and checking in descending order for a disk with any room for work files. If SORT selects a drive with insufficient space, it will abort later. Specifying a work drive with plenty of free space can avoid this occurrence.

<seq> . . . . . Non-ASCII collating sequence file. Specifies a file containing the character collating sequence to be used for the sort. If not used, ASCII sequence will be assumed.

SORT option parameters:

f . . . . . Format of output file. Codes recognized are I, C, and N.

The I option specifies a blocked output file, that is one record per sector. Refer to the previous section of this chapter for a discussion of text file formats.

The C option specifies a record compressed output file and may be used regardless of the input file format. The default output file format is record compressed, so the C option is never required.

The C and I options are mutually exclusive.

The N option specifies no space compression of the output file. By default, SORT writes the output file with the same compression format as the input file. That is, if the input file is space compressed, the output file will be also; if the input file has no space compression, the output file will not be space compressed. If the N option is used the output file will have no space compression even if the input file is space compressed.

o . . . . . Collating sequence order. The options entered may be A (ascending) or D (descending). The default sequence is ascending. If some keys are to be sorted in ascending order and other keys in descending order, the "o_n" specification described below should preceed each key whose order differs from the order of the key preceeding it. However, if all keys are to be ordered in the same sequence, this option need only be specified once.

r . . . . . Record format. This parameter specifies a special output record format: L for Limited output file format, T for Tag file output, or K for Keytag file output. The default value is no special output record format, so that the records in the output file will be exact copies (full image records) of the records in the input file.

Normally the sort transfers all of the records of the input file to the output file. It is possible, not only to transfer part of each record, but to select only certain records or to include constant literals in each record as well. Including the 'L' parameter in the list of parameters will cause another question to be asked wherein you may specify the limitations and constants. See the section on Limited Output Format Option.

By entering the 'T' character, an output file is generated which consists only of binary record number and buffer byte pointers to the input file records. See the section on Tag File Output Format Option.

By entering the 'K' character, a standard text format output file is generated which consists of records containing a 5 byte user logical record number, a 3 byte buffer address, and the key. These records are space-compressed and have trailing spaces truncated. See the section on Keytag File Output Format Option.

h . . . . . Hardcopy output. Entering options H or X causes the output of SORT to be listed on a printer. H specifies a local printer for output; X specifies a servo printer.

When hardcopy output is used, the output file specification becomes optional. If an output file was specified, SORT will write an output disk file as well as printing its output. If no output file was specified, SORT will produce only printed output. Except when performing hardcopy output the output file specification must be entered on the command line.

If hardcopy output is specified, limited output (L option under "Record format" above) must also be specified. For more information see the section on Hardcopy Output Option.

gnnntc. . . . . Primary/Secondary SORT specification. These parameters are all a single option and must be entered completely and in the order indicated. The separate parameters of the option are listed below.

g . . . . . Group indicator. Specifies that the input file consists of primary and secondary records and specifies which group is to be sorted. The character specified may be P for primary or S for secondary. There is no default value.

In a file with primary and secondary records, a string of records with a primary record as the first record and secondary records following it is

CHAPTER 43. SUR COMMAND

43.1 Purpose

The use of the SUR (Subdirectory Utility Routine) command allows the user to logically partition the directory on a given disk into several smaller subdirectories. Each such subdirectory can then contain zero or more files, up to the combined maximum of 256 files per logical drive. The reasons for such a capability are readily apparent. When a specific disk is used for more than one purpose, some inconveniences turn up. Assume for a moment that a user has a disk which he is using for program generation on each of two more or less unrelated projects. When he uses the CAT command, for instance, he will normally see a whole range of files, some of which are not related to the project he may be currently interested in. Or, he may begin editing a new file on the disk, only to find that another user of the same disk may have already had a file of that name. Without the DOS subdirectory facility, it is not permitted to have two files on a given logical drive with the same name.

43.2 Use

The SUR command is parameterized as follows:

SUR [<name> [/<function>]][:DR<n>][,<new name>]

The function performed by SUR is determined by the absence or value of the <function> field and the name field, as described below. If a specific drive is mentioned then that function is performed on only that drive, otherwise, it is performed on all drives. The only exception to this is remote ARC volumes. The only function permitted on a volume located at an ARC file processor is the display function. An attempt to perform any other function on one of these volumes will result in the message

DRIVE nn IS A REMOTE ARC VOLUME

being displayed and the function will not be performed on that volume.

43.2.1 Establishing a "Current Subdirectory"

If the function field is not given, SUR establishes the named subdirectory as the current subdirectory on all drives on which the named subdirectory exists. If the named subdirectory does not exist on one or more drives, the current subdirectory on any such drives is unaffected. If a specific drive is mentioned, then only the current subdirectory on the specified drive is subject to change.

43.2.2 Creating a Subdirectory

If the function field is /NEW, SUR creates the named subdirectory on all drives on which the named subdirectory does not exist. The current subdirectory is not affected by the operation. If a specific drive is mentioned, then the named subdirectory is only created on the specified drive.

43.2.3 Deleting a Subdirectory

If the function field is /DEL, SUR deletes the named subdirectory on any drives on which the named subdirectory exists. If any files are in the named subdirectory, they are moved to subdirectory MAIN before the named subdirectory is deleted. If the subdirectory being deleted is the current subdirectory on that drive, the current subdirectory is also changed to MAIN. Subdirectories SYSTEM and MAIN cannot be deleted. If a specific drive is mentioned, then the named subdirectory is only deleted from the specified drive.

43.2.4 Renaming a Subdirectory

If the function field is /REN, SUR renames the named subdirectory on any drives on which the named subdirectory exists, to the name specified in the new subdirectory name field. If any files are in the named subdirectory, they will be in the subdirectory specified by the new subdirectory name field upon completion of the operation. Subdirectories SYSTEM and MAIN cannot be renamed. If a specific drive is mentioned, then the name of the named subdirectory is changed only on that specified drive.

43.2.5 Displaying Subdirectories

If the subdirectory name field is not given, SUR displays the names of all subdirectories on all on-line drives. The format of the listing is similar to that provided for file names by the CAT command. The number in parentheses to the right of each subdirectory name is the subdirectory number associated with that name (in octal); an asterisk indicates the current subdirectory on each drive. If a specific drive is mentioned, then only the subdirectories present on the specified drive are displayed.

43.3 About Subdirectories

Each subdirectory on a disk has a unique name. Two subdirectories always exist on all drives; these are called SYSTEM and MAIN. A maximum of 31 subdirectories can exist on any volume. Since two are already used (SYSTEM and MAIN), there are 29 subdirectories available for user specification. The names for these subdirectories are assigned by the user as he establishes them, and follow the same rules as for any standard DOS file name. As a subdirectory is created, the name specified by the user is related to a unique number which is referred to as the subdirectory number. The relationship between subdirectory names and subdirectory numbers is similar to the relationship between DOS file names and physical file numbers. A given subdirectory may have different numbers on different drives, even though the subdirectory name is the same.

It is important to realize that subdirectories are not a way of getting more than 256 files on a drive. This they cannot do. The thing that subdirectories are good for is partitioning the directory and restricting the scope of a file name. This allows several files of the same name to exist on one disk at the same time, without causing the DOS to become confused as to which is the one to be referenced at any time. The way the DOS achieves this is that each of the files is in a "different subdirectory", and hence is uniquely identified even though the name and extension may be identical.

43.3.1 Creation of Subdirectories

Subdirectories are created with the SUR command. All that is required is to specify a name for the proposed subdirectory and request its creation. Creation of a subdirectory does not actually result in any real change to the directory on disk; all it does is cause the specified name to be entered into a table in SYSTEM7/SYS which relates each subdirectory name with its subdirectory number. The user is allowed to specify which drive he wishes to create the subdirectory on; if he does not indicate a specific drive, the named subdirectory is placed onto all on-line drives if possible.

43.3.2 Deletion of Subdirectories

Subdirectories are deleted with the SUR command. The user specifies the name of the subdirectory he wishes to remove and requests its deletion. Deletion of a subdirectory does not result in KILLing the files within the range of that subdirectory. If a subdirectory to be deleted contains one or more files, the files are first moved from that subdirectory to the one called MAIN before the named subdirectory is deleted. It is important to note that it is possible to get more than one file with the same name and extension in subdirectory MAIN because no check for a matching file is made. The user is allowed to specify from which drives the subdirectory is to be deleted; if he does not indicate a specific drive, the named subdirectory is deleted from all on-line drives on which it appears. Subdirectories may not be deleted while PS is running.

43.3.3 Being "in a Subdirectory"

The user can define at any time which of the subdirectories on each of his disks contain the current files he is interested in. This is done with the SUR command by specifying the name of the subdirectory containing the files of current interest. This action causes him to be placed "into" the named subdirectory on the drive specified. (If no specific drive is mentioned, he will be placed "into" the subdirectory specified on all on-line drives containing a subdirectory with the given name). It is appropriate to point out that the current subdirectory on each drive need not have the same name; for example, the user could easily be in subdirectory PROGRAMS on drive zero and in subdirectory DATABASE on drive one at the same time. This becomes even more complex when running under ARC. It is perfectly legal for a user to MOUNT the same physical volume as several

logical volumes, each in a different subdirectory. Thus drive zero could be volume ABC in subdirectory A and drive one could be volume ABC in subdirectory B.

Once in a specific subdirectory on a drive, that state does not normally change until the user requests being placed into a different subdirectory (again via the SUR command) or re-boots the DOS. Booting the DOS causes the user to be placed into the subdirectory named SYSTEM on all drives.

43.3.4 Scope of a File Name

When a program accesses a file under DOS, it tells DOS the name and extension of the file it is looking for and either indicates one specific drive which the DOS is to search for the file, or requests that the DOS look on all on-line drives. In order for the DOS to "find" the given file, the DOS must find a file whose name and extension exactly match the ones specified by the requesting program. If the current subdirectory (for that drive) is not SYSTEM, then the file must be in the current subdirectory or in SYSTEM to be found. If no such file can be found, the DOS returns indicating that the specified file cannot be found and therefore probably does not exist.

Therefore the scope of a file name can be more or less defined via the following: when a user is in subdirectory X on drive Y, files can be "seen" by his program only if they are in either subdirectory X or subdirectory SYSTEM. Files in any other subdirectory will not appear to exist.

The entire above procedure does not apply when OPENing a file by PFN since subdirectories are not checked in this case.

43.3.5 About Subdirectory SYSTEM

It has been shown that files in the subdirectory named SYSTEM are special in that they can be accessed regardless of which subdirectory the user is "in" on a specific drive. Likewise, a special situation also occurs when the user is "in" the subdirectory named SYSTEM. When the subdirectory named SYSTEM is the current subdirectory on a given drive, all files on that drive are accessible regardless of which subdirectory they themselves are actually in.

A little caution must be used when a user is in subdirectory SYSTEM on a disk with multiple files of the same name and

extension. The caution is that, although each of the files is still associated with one and only one subdirectory, all of the files on a disk are available when the user is "in" the SYSTEM subdirectory. The result is that in this situation, one of the files of the desired name and extension will be referenced; which one is referenced is, however, undefined. Therefore, good practice dictates that if a user has more than one file with the same name and extension on some drive, that he make a point of always knowing which subdirectory he is in (and that it is not SYSTEM) if it matters to him which of his files he references.

43.3.6 Files vs. the User Being "in a Subdirectory"

It is important not to confuse the two distinct concepts of a file being in a subdirectory as opposed to that of [a user] "being in a subdirectory".

A file being in a specific subdirectory is a way of saying that the file can be accessed only when the current subdirectory is either that specific subdirectory or SYSTEM. This relationship, that of a file being in a specific subdirectory, is retained more or less permanently; if a file is placed in subdirectory SUBDIR1 today on a disk, the disk can be removed and stored on a shelf; if tomorrow the disk is taken down from the shelf and re-mounted, that file will still be in subdirectory SUBDIR1.

A user being in a specific subdirectory is a way of saying that the subdirectory in question is "the current subdirectory" on one or more logical drives. The "current subdirectory" on a drive is less permanent and reflects the use of the SUR command since the previous time the DOS was bootstrapped.

43.3.7 Getting a File into a Subdirectory

In general, there are two ways to get a file into a given subdirectory. The easiest and probably most common of these is automatic. Whenever a file is created, it is always placed into the current subdirectory on the drive on which it is created.

Once a file has been thus created, it can be moved between subdirectories with the NAME command. The NAME command can take a file within the scope of the current subdirectory and put it into the current subdirectory if it is not already (which is useful if either the source or destination subdirectory is SYSTEM) or can

place it into any other subdirectory the user might wish to put it into.

considered one block, or group, of records.

When the file is sorted on primary records the output file has the blocks of records re-ordered so that the primary records are in the sorted sequence; no change is made in the sequence of the secondary records following each primary record.

When the file is sorted on secondary records and the first key specified is in ascending sequence, the output file has the blocks of records in the same order as in the input file, and the secondary records within each block in the sorted sequences.

When the file is sorted on secondary records and the first key specified is in descending sequence, the output file has the blocks of records in reversed order as the input file, and the secondary records within each block in the sorted sequence.

SORT has no provision for the sorting of primary and secondary records in the same SORT run.

nnn . . . . . Numeric position of primary/secondary flag. This parameter specifies the character position for the character (the 'c' parameter) indicating whether the record is a primary or secondary record. The number must be within the range 1 to 249. It is not necessary to enter leading zeroes to pad the number out to three digits, but do not use any leading blanks or an error will result.

t . . . . . Type of evaluation. This parameter specifies equivalence or inequivalence of the group indicator character; that is, whether the character in the record should be equal to or not equal to the character specified. The actual character entered is '=' for equal or '#' for not equal. There is no default value, either '=' or '#' must be entered.

If '=' is given then if the character in the nnnth position of an input file record is equal to the group indicator character (indicated by 'c' below) then the record is a member of the specified sort group (indicated by 'g' above). Otherwise, it is

not a member of the specified group.

c . . . . . Character, group indicator. This parameter specifies the actual test character for determination of a record's membership in the sort group. The actual character entered may be any 8-bit value except 015 (ENTER value). There is no default character: the character immediately following the 't' parameter is taken to be the 'c' parameter (except 015, which would be an error).

k_1. . . . . Sort Key specification, sss-eee. If no key is specified, the SORT will assume 1-10; that is, the first ten characters of the record.
sss is the starting key position.
eee is the ending key position.
The key is limited to 118 characters total, summing the length of all specificatons.

o_n. . . . . Order for the nth sort key. May be A (ascending) or D (descending) as described for the "o" parameter above. If omitted the order used on the previous key is assumed.

k_n. . . . . The nth sort key specification (sss-eee). The maximum number of keys is simply the number that can be typed on the command line without exceeding the line length.

42.4.2 Keys: Overlapping and in Backwards Order

The key specification need not be only forward. A specification of 17-12 will cause the 6 delimited characters to be a key but in the order of 17,16,15,14,13,12. This is extremely valuable, clearly, in data which has the most significant digit or character last.

Key specifications may also be overlapping: 1-20,30-15 overlaps 15 to 20. When this occurs, the system will optimize the sort and save time over re-sorting on those columns again.

42.4.3 Collating Sequence File

By specifying a sequence file, the user may substitute any collating sequence for the standard ASCII character set. The sequence file may have any name, but the extension must be "/SEQ" (SEQ is the default extension). If the disk drive number on which the file resides is omitted, SORT defaults to the same drive from which the SORT itself was loaded. This table may be supplied by the user but must meet certain requirements to be loaded:

1. It must be an absolute object file.
2. It must begin loading at location 027400.
3. The first eleven bytes must contain the file name and the extension must be SEQ. (Full 8 - character file name with trailing blanks, then extension.)
4. The table itself must begin loading at location 027400 and occupy 256 bytes (overstoring the file name described in 3). For instance, the source for the EBCDIC sequence file begins:

```
SET     027400
DC      'EBCDIC   SEQ'
SET     027400
DC      0,1,2,3,4,5,6,7,
.
.
.
```

5. If the file is not found on the specified disk drive the following message is displayed:

   SEQUENCE FILE NOT FOUND

6. If the file is found but is not an absolute object file the following message is displayed:

   SEQUENCE FILE FORMAT ERROR A

7. If the file format appears valid, the file will be loaded using DOS routine LOADX$. LOADX$ will return an error code if the load is unsuccessful. The following display will notify the user of the error:

   SEQUENCE FILE FORMAT ERROR n

   where n=0 if file does not exist
   1 if disk drive is off-line
   2 if directory parity fault

3 if RIB parity fault
4 if file parity fault
5 if off end of physical file
6 if record of illegal format

42.4.4 Ascending and Descending sequences

Changing the collating sequence from ascending to descending is the same as 'reversing' the file, or placing the last first, etc. Sorting a telephone directory in ascending sequence on name produces the familiar order. Should it be sorted in descending sequence, then Mr. Zyk would be first and Ms. Aardvark would be last. The order of collation, when alphabetic, numeric, and punctuation characters all can occur in a column together, follows the character set order. The sequence may be specified for each sort key. However, it need not be specified if it is the same as the key which preceeds it. Therefore, it is possible to sort portions of the key in ascending order and portions in descending order.

42.4.5 Input/output File Format Options

SORT accesses each file sequentially. Due to the techniques used in the Datapoint standard file structure, the sequential reading technique will provide SORT with all of the records in the file whether the file was originally blocked or sequential. Therefore, the file format options only allow specification of the output file's format.

If the input file is blocked, that is one logical record or string per physical disk record, then you have a choice of output formats (f option). If 'I' is chosen, that is blocked, then each output disk record will contain an exact copy of the appropriate input file record. If 'I' is not specified, then the input file, reordered, will be reblocked and appear, generally much more compactly, in the output file in record-compressed sequential format.

If the input file is sequential in its original format, then there is only one choice for the output format; the output file format for a sort on an input file which is sequential must be sequential.

42.4.6 Limited Output Format Option

In many cases, especially when making reports, directories etc. from the data base, it isn't necessary to have the entire record transferred from the input file to the output file during a sort. For instance, an entire personnel data base can be sorted by name to produce an internal company telephone directory. However, it is obvious that all that is needed is the name and telephone number, not all the other payroll information. Therefore, SORT permits transferring only that part of the data base desired.

If the 'L' option is entered in the SORT command line, the operator will be prompted for a second input line when the program displays a message:

LIMITED OUTPUT FILE FORMAT:

The cursor will be left flashing on the bottom line of the CRT and the program will wait while the operator enters the limited output specification line. This line may contain any number of specifiers as defined below, so long as it fits on a single screen entry line. The limited output specificatin must consist of at least one valid character output specification.

The following is the generalized statement format for the limited output specification:

{(sss[-eee]¦*¦'qqq')[/(P¦nnntc[(&¦+)nnntc[...]])]}[,{<limspec>}...][-]

Different items within parentheses are separated by a vertical bar (¦); only one item within parentheses may be specified in a limitation specification item. The collection of items within the braces ({}) constitute a limitation specification item. As many limitation specification items may be entered as will fit on a single entry line. The upper-case letter P and the special characters hyphen (-), asterisk (*), single quote ('), slash (/), plus (+), and ampersand (&) represent actual literal characters entered in the limitation specification line. The lower-case letters represent different values which will be entered on the limited output specification line.

The following list defines the parameters which can be specified:

Parameters to specify characters output:

sss[-eee]. . . Delimited character output from input record. Causes the characters from the specified columns of the input record to be copied to the output record.
sss is the starting column position.
eee is the ending column position.
The eee specification is optional; if not specified only the single character at column sss will be copied.
The column numbers specified must be in the range of 1 to 249.

*. . . . . ASCII TAG output. This parameter specifies that an ASCII pointer to the input record will appear in the output record. The ASCII pointer is a text numeric field with a 5-digit logical record number followed by a 3-digit byte pointer. The logical record number references the user LRN in which the text record begins; this number can be used for DATASHARE random access or other similar applications. The byte pointer points to the first byte of the text record within the specified LRN; this number can be used to tab within DATASHARE random access or other similar applications. The maximum value for the logical record number is 65,535; the maximum value for the byte pointer is 250. If the 'I' format option was specified in the command line option field, no byte pointer will be written. When the input file is in 'I' format the logical record number alone is sufficient to identify the text records, since each logical record always then begins in byte one.

'qqq'. . . . . Quoted character string. This parameter specifies an actual string of quoted characters that is to be copied into the output record. The quoting symbol is the single quote mark, or apostrophe ('). The string may include any characters except an apostrophe or an 015, and must be less than 90 characters long.

Parameters specifying special conditions for output. If used, these parameters must immediately follow one of the character output parameters. A slash (/) separates the

conditional parameter from the character output parameter to which it applies.

P. . . . . Primary record to be source. This parameter specifies that the information specified by the preceeding character output parameter is to be extracted from the primary record for the current record block, rather than the present (secondary) record. This parameter has no effect when an output record is being generated from a primary record.

&. . . . . Logical AND for multiple nnntc.

+. . . . . Logical OR for multiple nnntc.

nnntc. . . . . Conditional output dependent on character evaluation. The nnntc parameters constitute a single item and must all be specified in the correct sequence if the option is used. As indicated in the prototype specification line, multiple character evaluation specifications may be entered following a single character output parameter, separated from each other by ampersands (&) or plusses (+). If such multiple parameters are used the specified characters will be output if the left-to-right significant logical expression is satisfied.

nnn. . . . . Numeric position of evaluation character. This parameter specifies the character position for the character (the 'c' parameter below) indicating whether the information specified by the prior set of character output positions is to be copied from the input record to the output record. The number must fall in the range 1 to 249.

t. . . . . Type of evaluation. This parameter specifies the equivalence or inequivalence of the evaluation character; that is, whether the character in the input record should be equal to or not equal to the evaluation charater. The actual character entered is '=' for equal or '#' for not equal. If the evaluation is satisfied, then the information specified by the prior set of character output positions

will be copied to the output record.

c. . . . . Evaluation character. This parameter specifies the actual test character for record evaluation. The actual character entered may be any character except 015.

-. . . . . Line continuation. Limited output specifications can be continued to another line.

The limited output specification can specify that only a portion of each input record is to be transfered to the output file. Should the response "1-10" be given to the limited output format request, only the first ten characters of each record will be transferred to the output file. A specification of "1-10,50-70" would transfer thirty-one characters from each record of the input file to the output file. The eleventh character in the output record would be the fiftieth character of the input record, and so on.

To permit even more utility in report generation, SORT allows inclusion of constants in the output record that did not occur in the input record. For instance, assume that the personnel data base was a full record of about 240 characters and that the employee's name appears in columns 80 to 110 and his telephone number was in columns 171 to 180. To make a telephone directory in alphabetical order, one could answer the following to the limited output file format request:

```
80-110,' - ',171-180
```

This specification would put out the name followed by one space, a hyphen, one more space and the number. Any number of input file fields and constants can be placed in the output file up to the limit of the line on which the specification is typed.

Often not every record of the input file is needed in the output file. Limited output allows selection of parts of records from the input file, based on character evaluation on character position. For example, if a primary/secondary file is being sorted and only the primary records are desired in the output file, the command could appear as:

```
SORT INFILE,OUTFILE;LP1=*,2-10
```

```
LIMITED OUTPUT FILE FORMAT:      <--(program displays)
1-85/1=*
```

Columns 1-85 of the input record will be written to the output file if column 1 is an *.

Limited output can be used to make more complex selections. If it is desired to output records containing a 0 in column 5 OR a 1 in column 6, the command would be:

```
SORT INFILE,OUTFILE;L5-8,12-15
LIMITED OUTPUT FILE FORMAT:      <--(program displays)
1-85/5=0+6=1
```

To output only records containing a 0 in column 5 AND a 1 in column 6 the command would be:

```
SORT INFILE,OUTFILE;L5-8,12-15
LIMITED OUTPUT FILE FORMAT:      <--(program displays)
1-85/5=0&6=1
```

There is no relationship between the primary/secondary specification on the command line and the conditional output specification on the limited output format line.

Also note that the output file may require proportionally less room than the input file when limited. Often this fact can be put to use when the disk file space is nearly exhausted and a sort is required.

42.4.7 TAG File Output Format Option

For some applications it is useful to have a data file sorted into several different sequences. However, to have several copies of a file on disk merely to have it in different sequences consumes a lot of disk space, and indeed if the file is a very large file many copies of it may not fit onto one or even four disk packs.

This problem could be avoided if there were a way to index into the one main file in any of several different sequences. The index pointers could exist as a file, and the index entry for each record in the main file would only have to be three bytes long: two bytes for the LRN (Logical Record Number) and one byte for the BUFPTR (Buffer Pointer, a pointer to the beginning of the actual desired record within the disk physical buffer).

SORT provides for the generation of such an indexing file, a TAG file, by the 'T' variation of the 'r' option. A TAG file may be generated for either a sequential or blocked file, and will have the same format for either file. The format of a TAG file is simple:

1. For each record in the input file, the TAG file will have a three byte binary pointer to the first byte of the record.

2. The format of the pointer is:
   Byte 1: MSPLRN (Most Significant Portion of LRN),
   Byte 2: LSPLRN (Least Significant Portion of LRN),
   Byte 3: BUFPTR (Buffer Pointer).

3. The three-byte binary pointers are blocked 83 to a physical disk record.

4. The Physical-End-Of-Record mark is an 003 and the rest 000's.

5. The End-Of-File mark is: beginning at the first byte in the physical record, six 000's, one 003, and the rest 000's.

TAG files may be used by assembly language programs or by RPG II (as Record Address files).

For users writing their own Assembly language code to use a TAG file, it is important to know that the MSPLRN and LSPLRN are together a 16-bit binary pointer to the DOS LRN of the input file, as opposed to the User LRN. The difference is this: The DOS LRN of a file points to the actual Nth record (starting with zero, the primary RIB) in the file, whereas the User LRN of a file points to the Nth data record (starting with the zeroth data record) in the file. Thus a DOS LRN of zero points to the very first record of the file, which is the master copy of the RIB; a DOS LRN of one points to the second record of the file, which is the RIB copy; a DOS LRN of two points to the third record of the file, which is the first data record of the file and User LRN zero. The LRN given in the TAG file can NOT be used with the POSIT$ routine unless it is biased by -2. It is much easier to simply place the LRN from the TAG file directly into the LFT entry for the file that is indexed.

The case with the buffer pointer byte is similar to the LRN pointer bytes. The buffer pointer byte from the tag file is the DOS buffer pointer as opposed to the User buffer pointer. The difference is that the DOS buffer pointer points to the actual Nth byte of a disk buffer (starting with zero), whereas the User buffer pointer points to the Nth data byte in the disk buffer.

The beginning (zeroth) data byte in the buffer is the fourth byte in the buffer; the first three bytes are reserved for the DOS. Thus, a DOS BUFPTR of zero points to the very first byte in the buffer, which is the PFN (Physical File Number) of the file; a DOS BUFPTR of one points to the second byte in the buffer, which is the DOS LSPLRN; a DOS BUFPTR of two points to the third byte in the buffer, which is the DOS MSPLRN; a DOS BUFPTR of three points to the fourth byte of the buffer, which if the first data byte and User BUFPTR zero. The BUFPTR given in the TAG file can NOT be used with the GETR$ or PUTR$ routines unless it is biased by -3. It is much easier to simply place the BUFPTR from the TAG file directly into the LFT entry for the file that is indexed.

If a TAG file is generated when the 'P' option is specified then TAG file pointers will be generated only to the Primary records in the input file.

If a TAG file is generated when the 'S' option is specified then TAG file pointers will be generated that point to each Primary record of the input file (in their original sequence) each primary tag being followed by pointers to the Secondary records in the record block in their sorted sequence.

When a TAG file is generated for 'P' or 'S' sorts, no indication is given in the TAG file pointer as to whether the pointer points to a primary or a secondary record; it is up to the user's program to check the records in the indexed file to determine when a record block begins or ends.

42.4.8 Keytag File Output Format Option

Requesting a Keytag file output will cause a file (default extension "TXT") to be created. This EDIT-compatible text file contains the record pointers and the key. The record pointers (first 8 bytes of the record) consist of a 5 byte logical record number (range 0 to 65,535) and a 3 byte buffer address. The record number is the user logical record number, that is, zero points to the first data sector. Therefore, the user logical record number, converted to binary, may be used with the POSIT$ routine. The buffer address is the buffer pointer, that is, one points to the first data byte in a sector. It may be biased by 2 and placed directly into the Logical File Table, or if biased by -1, used by the GETR$ routine. This Keytag file output is the Keytag file used by INDEX.

If a sequence file (for example, EBCDIC/SEQ) is used, the key produced by this option will be translated to that sequence. If

the un-translated key is desired, a Keytag file may be created (slower) by requesting ASCII TAG output from the Limited Output Format Option.

42.4.9 HARDCOPY Output Option

Many times it is desired to have a hardcopy (printed) output from a SORT instead of or in addition to the creation of a disk output file. This can be easily accomplished with SORT by specifying the 'H' or 'X' option along with the 'L' (Limited Output) option. The hardcopy option is essentially an expansion of the 'L' option because disk data files are almost never suitable for full image output to a printer; decimal points need to be inserted into dollar and cents amounts, dashes need to be inserted into part numbers, and spaces need to be placed between dollar amounts and part numbers to columnate the data, and so on. If it is desired to list output records in full image format, it is only neccessary to give:

```
1-n
```

(where n is the maximum printable character on printer) as the limited output string specification.

Sort will not send a line of over 132 characters to a printer. If the limited output specification designates a longer output record, then the full specified formatting will be applied to the disk output file (if any), but only the first 132 characters of the record will be printed.

If the following special characters are imbedded in the output record, they will be interpreted as indicated:

```
015 = End-Of-Record and Carriage-Return/Line Feed.
012 = Line Feed.
014 = Form Feed.
```

The 'H' option specifies output to a local printer. The 'X' option specifies output to a servo printer. If the selected printer is not on-line then SORT will pause during final merge and display the message PRINTER OFF-LINE. While this message is displayed the program may be terminated by pressing the KEYBOARD key. Printing will commence and program operation will continue when the correct printer is brought on-line and ready.

42.4.10 Primary/Secondary Sorting Considerations

If the 'P' (Primary) or 'S' (Secondary) SORT option is used then the input file must have a PSPSPS.... format in order for SORT to work as expected, where P is one primary record and S is one or more secondary records. The first record in the file should always be a primary record, and the last record should be a secondary record. There should always be at least one secondary record following each primary record. Tertiary and further level records cannot be accommodated by SORT.

In some cases it may be possible to successfully sort a file using the 'P' or 'S' options even if the file does not faithfully follow the above rules, but problems will likely show up if such a sort is attempted. For example, if a file has the format PPPPSPSPS..., and a sort is done using the 'S' option, the output file will not contain the first three primary records at all. This case occurs because when sorting using the 'S' option, pointers are generated for only the secondary records, prefixed by a pointer to the record preceeding the first secondary record of a record block. Since no secondary pointers were ever generated for the first three primary records, they are simply lost.

42.4.11 SORT Work Files

SORT uses two scratch files during its operation: *SORTKEY/SYS and *SORTMRG/SYS. The first character of these file names will change when SORT is run under Datapoint's Partition Supervisor (PS). When PS is active the partition identifier (a one-digit number unique to each partition) is placed in the file name in place of the asterisk, so there is no danger of simultaneous SORTs trying to use the same scratch files. SORT always uses the same names for these files, only the drive on which they are placed can be affected by the operator (by means of the <drv> field on the command line).

SORT will always build the *SORTKEY/SYS file, since it holds the sort key trains generated as the first step of sorting. If more than one sort key train is built, SORT will also create the *SORTMRG/SYS file to merge the trains. Normally the work files will be deleted when SORT terminates, but if the SORT aborts or is interrupted for some reason the work files will be left on disk. If this happens they should be deleted using KILL by PFN, since the asterisk is not a normal part of a file name and cannot be scanned as such by the command interpreter.

42.5 Disk space requirements

A formula for determining the room in physical disk records that will be required for the SORT work files is:

$$R = \frac{2N(L+P+3)}{S} + 4T$$

where: R = Room in physical disk records (sectors) required on disk.
N = Number of logical records in input file for which keys will be generated:
= number of records in file if not sorting on 'P' or 'S'.
= number of primary records in file if sorting on 'P'.
= number of secondary records in file if sorting on 'S'.
L = Length of the sort key in bytes.
P = 3 if sorting on secondary records,
0 if not sorting on secondary records.
T = number of sort key trains.
S = bytes per block of physical space available to the user (nominally 253 bytes)

The value of T can be computed approximately, as:

$$T = \frac{N(L+P+3)}{(M-8000)/2}$$

Where M is the available processor memory size expressed in decimal.

42.6 LINK into SORT from programs

There are three ways in which a SORT can be initiated:

1. From the keyboard via the DOS COMMAND HANDLER;
2. By using the DOS CHAIN command;
3. By loading and linking to SORT/CMD from an assembly language program.

Datashare users can invoke SORT by using the rollout facility to start or continue a chain (see CHAIN and the DATASHARE User's Guide for more details).

The following detailed information is provided for users writing system-level programs in assembler language, since Datapoint does not release a source listing of the SORT program. Normal usage of SORT requires no knowledge of the following information.

Sort reserves for the user a nominal amount of storage normally occupied by the DOS DEBUG$ routine. The specific memory locations saved are 06144 through 06377. This permits the user to partially overlay his program with the SORT utility and regain control at the completion of the sort. Additionally, the next page of storage, 06400-06777, is available to the user if full image output records are to be generated. The DOS interrupt handler is disabled during the sort but is re-enabled upon completion of the sort. Of course, if the user has a foreground process running before and after the sort, the process must be controlled from within the memory not used by SORT, or when foreground is re-enabled it will vector to whatever SORT left in memory.

NOTE: New information for DOS 2.5. SORT now uses DOS Functions. The DOS Function loader must be intact when SORT is invoked. SORT itself no longer overstores the loader so DOS Functions may be used immediately following return from SORT. If the hardcopy output option of SORT is used, user forground processes cannot be left active during the SORT. The local printer driver may use interrupt slot 3 if it needs to wait for the printer to become ready. The servo printer driver always uses interrupt slot 3. If hardcopy output is used from SORT, the normal DOS interrupt scheduler must be available and there must be no active interrupt vectors.

The steps to call SORT from an assembler program are as follows:

1. Close files 1, 2, and 3 if open.
2. Set MCR$ (01400-01543) with the command string terminated by a 015.
3. Load the SORT utility.
4. PUSH the stack.
5. Point HL to a parameter table with the format:

```
PTABLE  DA    LIMSTG
        DA    HEDING
        DA    EXITAD
```

6. RETURN

Where:

LIMSTG = the limited output specification string, terminated by a 015. If there is to be no limitation output specification, put 0. If there is a LIMSTG, it must exist entirely within the range 06144-06377. The LIMSTG must be exactly the characters as they would be entered from the keyboard. Examples follow.

HEDING = the hardcopy heading string, terminated by a 015. If there is to be no hardcopy output, put 0. If there is a hardcopy heading string, it must exist entirely within the range 06144-06377. The HEDING must be exactly the characters as they would be entered from the keyboard. Examples follow.

EXITAD = the first memory location to be executed upon successful completion of the sort. If the sort is to return to the DOS upon completion, put 0. If there is a specific exit address, it must exist within the range 06144-06377. Normally, the instructions at the exit address will load and run the program to be run after the sort, or will re-load a control program of the user's own control system.

A simple example of loading and running sort from an assembler program would be:

```
                .
                .
 1.SRTCMD   DC    'SORT INFILE,OUTFILE',015 SORT CMD STRING
 2.SRTNAM   DC    'SORT   CMD'       NAME OF SORT UTILITY ON DISK
 3.PTABLE   DA    0                  NO LIMITATION STRING
 4.         DA    0                  NO HARDCOPY HEADING
 5.         DA    0                  NO SPECIAL EXIT ADDRESS
                .
                .
                .
 6.RUNSRT   LC    SRTNAM-SRTCMD      MOVE THE SORT COMMAND STRING
 7.         DE    MCR$               TO MCR$
 8.         HL    SRTCMD
 9.         CALL  BLKTFR
10.         LC    -1                 LOAD THE SORT UTILITY
11.         DE    SRTNAM
12.         CALL  LOAD$
13.         PUSH                     PUSH THE SORT STARTING
                                     ADDRESS
14.         HL    PTABLE             POINT TO THE PARAMETER TABLE
15.         RET                      RUN SORT
                .
                .
```

The above sequence of instructions could be located anywhere in memory, except lines 13 thru 15 must obviously reside in a portion of memory from 06144 thru 06377 to avoid being overlayed when the SORT utility is loaded from disk. The above instructions exemplify the simplest possible case of linking to SORT, in that only the SORT command and an input file and an output file are specified, all other options are defaulted. The above instructions have the same effect as calling SORT by entering the line:

```
SORT INFILE,OUTFILE
```

to the DOS COMMAND HANDLER.

Here is a line-by-line explanation of the instructions:

Line 1 defines the SORT command string. This is accomplished by a simple DC statement of a quoted ASCII string followed by a 015. The quoted ASCII characters are exactly the same that would be keyed in to the DOS Command Handler if the sort were being initiated from the keyboard. The 015 is the string delimiter and is the same character that is placed after a string by the KEYIN$ routine when the "ENTER" key is depressed. The SORT command string can be up to 100 characters long including the 015 because

the MCR$ area is 100 bytes long. Note that this is nineteen characters more than can be specified from the keyboard.

Line 2 defines the name of the SORT utility main overlay. Notice that the complete name of the SORT given here must be exactly the name as listed in the DOS directory of files. The eleven ASCII characters in a file name specification include an eight character filename and a three character extension. Since the filename of SORT is only four characters, it must be followed by four spaces before the extension of "CMD" can be given.

Line 3 defines the beginning of the six-byte parameter table. The first two bytes of the parameter table specify the address of the beginning of the Limited Output Specification string. In this example there is to be no limited output specification string, so an address of 0 is given.

Line 4 defines the address of the beginning of the HARDCOPY HEADING string. In this example there is to be no hardcopy output, so an address of 0 is given.

Line 5 defines the address of the Exit Address, or the address to which the SORT is to exit when it is successfully completed. (If something goes wrong during the sort, exit is to the DOS.) In this example there is to be no special exit address, so an addess of 0 is given.

Line 6 begins the actual process of calling SORT from the program. Lines 6 thru 9 move the SRTCMD string from wherever it is in memory to the MCR$ area.

Line 10 specifies that SORT is to be loaded from wherever it is found in the disk drives that are on-line to the system. Refer to the chapter on System Routines if you are not familiar with the DOS LOAD$ routine.

Line 11 points to the name of the SORT utility main overlay in memory, given in SRTNAM, line 2.

Line 12 calls the DOS LOAD$ routine which finds the SORT main overlay program on disk and loads it into memory, leaving the starting address in HL.

Line 13 puts the starting address of SORT on the P-counter Stack.

Line 14 points to the Parameter Table, lines 3, 4, and 5. The way that SORT knows that it is being run by the DOS Command

Handler or by a user program is by comparing the values of the HL contents with the entry point of SORT. If the values are equal, as they are immediately following a LOAD$, then SORT asks for a Limited Output Specification string and a Hardcopy Heading string if they are specified in the SORT COMMAND string. If the values are not equal, then SORT checks the memory pointed by HL for the location of the Limited Output Specification string, the Hardcopy Heading string, and an Exit Address.

Line 15 effects the actual transfer of execution to the SORT utility. Since the starting address of the SORT was PUSHed onto the P-counter stack, a RETurn instruction JuMPs to the SORT starting address.

42.7 The Use of CHAIN with SORT

The reader should first familiarize himself with CHAIN by thoroughly reading the CHAIN Section.

CHAIN is a system whereby the operator of a Datapoint DOS may pre-define a procedure sequence of his own programs, system commands and utilities (including keyboard answers to questions requested by these programs) and have them called and sequentially executed by a single name. This feature is especially powerful when using SORT since there may be a repetitive sequence of routines with complex parameterizations which could make good use of simplification.

A Datashare program can link to SORT by executing a ROLLOUT instruction to a user-built CHAIN file which includes the SORT command line and, if specified, the Limited Output specification line and a Hardcopy Heading line, followed by the DSBACK program to re-load the Datashare.

42.7.1 Defining a Chain File for SORT

The author of a chain file only needs to remember that all questions that the system requests including those initiated by the executing programs must be answered from the chain file just as though they would be typed in from the keyboard.

For instance, the initiation of a sort

"SORT INFILE,OUTFILE;I3-42"

could be done through chain. To do this, use EDIT or BUILD to type in that exact sequence of characters into a file. Note that the file will, in this case, consist of a single line as typed above. The file can be any name, but for purposes of simplifying the explanation, it shall be referred to as "CHAINFIL". If "CHAINFIL" consists of that single line, and if the operator types the command "CHAIN CHAINFIL" to the DOS, the SORT specified above would be initiated. If the 'L' specification were included in the statement above, then SORT would ask for another line of information. In this case, the file "CHAINFIL" would have to have two lines in it with the first being the SORT command and the second being the limited output file format specification.

42.7.2 Naming a repetitive SORT procedure

Frequently there are sorts and printouts and other procedures which occur together and for which a name invoking the procedure would be a great simplification.

For instance, in the telephone directory example above, the process of sorting the file into a limited output file and then listing it on a local printer could be procedurized as follows:

```
SORT EMPFILE,TELFILE;L80-110
80-110,'  - ',171-180
LIST TELFILE;XL
TELEPHONE DIRECTORY FOR XXXXXXXXXX CORPORATION
```

Note that there are four statements. The first is the SORT command. The second is the answer to the limited format initiated by the 'L' in the SORT command. The third is the DOS LIST command with the specifiers of 'X' which says 'without line numbers' and the 'L' which means local printer. Then there is a fourth line which the LIST command requests - the heading. This question must also be answered in the chain file. If the above four statements were placed in a file by the Editor (or by any other means) and then CHAIN were invoked with that file specified, the result would be a printed telephone directory from the personnel files.

42.7.3 Using CHAIN to cause a merge

Consider a situation wherein a system has a master file called 'MASTER' and a file of records to be added, in sequence, to the master file called 'ADDFILE'. To merge these two files in sorted sequence at the end of each day would normally require a sequence of keyed in operations which are somewhat complicated and error prone. CHAIN can cause an effective MERGE and assign it a single name as follows:

```
SAPP MASTER,ADDFILE,MASTER
SORT MASTER,SCRATCH;1-20
KILL MASTER/TXT
Y
NAME SCRATCH/TXT,MASTER/TXT
```

Note that the procedure:
1) appends the ADDFILE to the MASTER file.
2) Sorts the extended MASTER file into a SCRATCH file.
3-5) Renames the SCRATCH file as the new MASTER file. Thus, it is apparent that a merge can be effectively achieved using SORT and by using chain to pre-define the procedure.

42.8 SORT Execution-Time Messages

This section describes the operator messages that SORT may display on the CRT screen during execution. Some of the messages are monitor messages to keep the operator informed of the progress of the program, while other messages are error messages.

DOS. VER. n.n SORT COMMAND - date

This message is the SORT sign-on.

SORT OVERLAY MISSING.

This message is displayed if the SORT/OV1 file is not on the same drive as the SORT/CMD file.

INPUT FILE REQUIRED.

This message is displayed if no filename was specified for the first file specification. This would happen if a command line such as:

SORT ,OUTFILE or SORT /TXT,OUTFILE

were entered.

BAD DEVICE SPECIFICATION.

This message is displayed if a drive specification in a file specification was not entered in a valid format.

OUTPUT FILE SAME AS INPUT.

This message is displayed if the filename and extension of the input file and the output file are the same, and the drive for each file is the same or not specified for both files.

INPUT FILE NOT FOUND.

This message is displayed if the INPUT file could not be found on any drive on-line to the system if no drive was specified, or on the drive given if a drive was specified. If no extension is supplied in the file specification an extension of TXT will be assumed.

KEY FILE SPECIFICATION ERROR.

This message is displayed if a FILENAME or EXTENSION is given for the KEY DRIVE specification.

KEY FILE DEVICE SPECIFICATION ERROR.

This message is displayed if the drive specification for the KEY file is not a valid drive spec.

SORT KEY FILE PLACED ON DRIVE #

This message is displayed if the KEY DRIVE was not specified on a multi-drive system. The message is to notify the operator of the location of the KEY file. The # stands for a valid drive number.

OPTION FIELD ERROR.

This message is displayed if a semicolon (;) is entered at the end of the SORT command line but is not followed by any option specifications.

OPTION SPECIFICATION DUPLICATION.

This message is displayed if a command line such as:

```
SORT INFILE,OUTFILE;DLA
```

were entered. The 'D' and 'A' options are both variations of the ORDER option, and obviously both cannot occur simultaneously.

HARDCOPY ONLY IF LIMITED OUTPUT SPECIFIED.

This message is displayed if the 'H' or 'X' option is specified but the 'L' option was not specified.

ILLEGAL HEADER SPECIFICATION.

This message is displayed if the 'P' or 'S' option is given but is immediately followed by the 015 byte -- the "ENTER" key.

ILLEGAL HEADER KEY EVALUATION.

This message is displayed if the character immediately following the 'PNNN' or 'SNNN' option is not '=' or '#'.

ILLEGAL SORT KEY SPECIFICATION.

This message is displayed if a key position of 0 or greater than 32,767 was specified, or if a key position was not terminated by "," or "-" or 015, or if a two-position key was not terminated by "," or 015.

SORT KEY TOO LONG.

This message is displayed if the total sort key is longer than 118 characters long.

OVERLAPPING SORT KEY SPECIFICATIONS---SORT OPTIMIZED.

This message is displayed if the same record positions were specified for more than one sort key group. SORT does not

repeat duplicate positions in sort key generation and thus saves processing and disk read/write time.

OVERLAPPING SORT AND HEADER KEYS---SORT OPTIMIZED.

This message is displayed if the same record position is specified as a sort key position and a header indication position. The position is removed as a sort key position and the key is thus shortened. The effect is as for the previous message.

LIMITED OUTPUT FILE FORMAT:

This message is displayed if SORT has accepted the SORT command line including all option specifications and if the 'L' option has been given. The operator must enter the limited output specification line.

NULL LIMITATION SPECIFICATION.

This message is displayed if the 'L' option was given but the limitation specification was only 015 -- the "ENTER" key. If the 'L' option is given then a non-empty limited output specification string must also be given.

INVALID LIMITATION SPECIFICATION.

This message is displayed if the limited output specification does not fit the syntax given in the section on Limited Output Format Option. Usually the fault is that a comma was not placed between option specification groups, or double quotes (") were used instead of single quotes (') .

ENTER THE HARDCOPY HEADING:

This message is displayed when the limited output specification has been accepted and if the 'H' or 'X' option was given. The operator must enter from 0 to 79 characters of information which will be printed at the top of each page printed during SORT output generation.

SEQUENCE FILE NAME REQUIRED

This message is displayed when the sequence file field is blank and the file specification fields have not been terminated with a semi-colon or an end of line designator.

SEQUENCE FILE NOT FOUND

This message is displayed when SORT requests the sequence file be OPENed and DOS cannot locate the file on the disk drive indicated. Note that if the drive is not specified, the drive on which the SORT/CMD resides is implied.

SEQUENCE FILE FORMAT ERROR A

This message is displayed when SORT determines that the sequence file specified is not an absolute object file.

SEQUENCE FILE FORMAT ERROR n

This message is displayed when SORT receives an error return from LOADX$ when an attempt is made to load the sequence file. The value of n may be 0-6 and is defined as follows:

0 If file does not exist
1 If disk drive is off-line
2 If directory parity error
3 If RIB parity fault
4 If file parity fault
5 If off end of physical file
6 If record of illegal format

LIMITATION SPECIFICATION OVERFLOW

This message indicates that limited output parameters entered require more memory (256 bytes) than allocated by SORT.

INTERNAL ERROR -- GET SYSTEM HELP !!!

This message indicates a probable hardware error occurred during a limited output string sort. SORT cannot continue executing.

The following messages may be displayed during sort initialization if SORT were linked to by an assembly language program:

INVALID LIMITATION STRING ADDRESS.

INVALID HARDCOPY HEADING STRING ADDRESS.

INVALID USER EXIT ADDRESS.

One of these messages is displayed if the corresponding entry in the parameter table linkage data was not either 0 or in the range 06144-06377 inclusive.

LFT ENTRIES 1->3 NOT CLOSED WHEN SORT ENTERED.

This message is displayed if the user left one of the logical files 1, 2, or 3 open upon linking to the SORT utility.

LIMITATION STRING MISSING.

This message is displayed if the 'L' option was given in the SORT command string but the pointer to the limited output format string in the parameter table linkage data was 0, indicating no limited output format string specified.

HARDCOPY HEADING STRING MISSING.

This message is displayed if the 'H' or 'X' option was given in the SORT command string but the pointer to the hardcopy heading string in the parameter table linkage data was 0, indicating no hardcopy heading string specified.

The following messages may be displayed after the SORT initialization is completed:

BUILDING SORT KEY TRAIN n.

This message is displayed when all parameter specifications have been accepted and SORT has started the extraction of the sort keys from records of the input file. The trains are

build in memory, sorted, and writted to the *SORTKEY/SYS file.

SORT KEY FILE OVERFLOW.

This message is displayed if there was not adequate room on the selected drive to hold the *SORTKEY/SYS file. If *SORTKEY/SYS file overflow occurs the file is deleted from the disk before the message is displayed.

NON-TEXT CHARACTER IN INPUT FILE. LRN nnnnn

MISSING EOS IN INPUT FILE. LRN nnnnn

INVALID EOF IN INPUT FILE. LRN nnnnn

One of the above messages is displayed if the corresponding error condition is found in the input file while building the sort key trains. A non-text character is an octal zero (eof character) found in the text string. The message "MISSING EOS ..." indicates that a sector has no 003 marking the logical end of sector. The message "INVALID EOF ..." indicates that a 000 was found in the first data byte of a sector, but the sector did not contain a complete text end of file mark. The LRN displayed is the User LRN containing the invalid text data, in octal. The error condition in the input file must be corrected before the file can be sorted.

NULL OUTPUT FILE.

This message is displayed if no sort key records were generated. A null output file (first record EOF) is prepared before SORT ends.

INTERMEDIATE MERGE PASS n, TRAIN n.

This message is displayed if more than sixteen sort key trains exist during a merge pass. The intermediate merge pass number is the Nth iteration of the merge process. The train number is the number of the train being output by the merge pass. If more than sixteen trains are output by an intermediate merge pass then at least one more intermediate merge pass will be required to merge those trains.

FINAL MERGE: SORT TRAIN n.

This message is displayed during the generation of the output file from the data in the sort key file and from the records in the INPUT file. The sort train number corresponds to the current state of progress as measured against the number of trains generated by the last intermediate merge pass.

MERGE FILE OVERFLOW

This message indicates not enough disk space is available for the merge file.

OUTPUT FILE OVERFLOW

This message indicates not enough disk space is available for the output file.

CHAPTER 44. UBOOT COMMAND

44.1 Purpose

The UBOOT command writes a DOS bootblock onto the cassette tape in the front tape deck. The resulting "boot" tape is used to initiate system loading when the processor is restarted.

44.2 Use

UBOOT is invoked by entering the command

UBOOT

The program verifies that there is a tape in the front cassette deck and asks the operator for permission to write on the tape. When the operator instructs the program to proceed it writes a DOS bootblock onto the cassette in the front deck. The UBOOT command then rereads the bootblock to insure that the cassette is good. In addition, the bootblock checks its own parity immediately upon loading and halts if it finds it has not been loaded properly.

After the boot tape has been written the program asks the operator whether to write another tape or to return to the operating system. The operator may insert another cassette and ask the program to write another boot tape as often as desired. The program terminates when the operator instructs it to do so.

If the machine halts upon booting repeatedly and other boot tapes work on the same machine, then the boot tape which causes the boot operation to halt is not a good tape and should be replaced.

44.3 UBOOT System Load Operation

The boot tape created by UBOOT reads an IPL (Initial Program Loader) block from disk. The IPL block then reads and executes the DOS bootblock (from disk). The IPL and bootblock are put on disk by DOSGEN and PUTIPL.

The UBOOT tape is capable of loading any version 2.3 or later

DOS from any type of disk. If there are multiple types of disks on your system, they will be scanned in the following order:

1. Mass storage disks (9390 first, then 9370/9374)
2. Cartridge disks
3. Floppy disks

Logical drive zero will be tested on each of the disks. If drive zero is off-line, depressing the "DISPLAY" key will cause a scan of ALL on-line drives. This means that if drive zero is "down", you can generally continue running. When a disk is found that contains a good IPL, it will be selected as the "BOOT DRIVE"; henceforth overlays will be loaded off it. Commands will also be loaded from the booted drive first (default).

CHAPTER 45. UTILITY/OVL

The DOS for processors using the 5500 instruction set (DOS.D, DOS.E, and DOS.G) include a file "UTILITY/OVL". This file contains memory resident copies of the DOS overlays and DOS Functions. These copies of the overlays reside in system RAM between 0160000 and 0167377. While the memory resident overlays are present, calls to DOS overlays (OPEN$, PREP$, and so on) or to DOS Functions will not require accessing the disks to load the routines. The result is improved system performance. The operating system automatically loads and initiates use of UTILITY/OVL whenever possible; programs needing system RAM for their own purposes disable use of the memory resident overlays.

Use of UTILITY/OVL is initiated by DOS Function 12. Usage is terminated by DOS Function 16.

CHAPTER 46. UTILITY/REL

The UTILITY/REL file contains a number of system utility routines in relocatable code format. The purpose of these routines is to simplify and standardize the use of common peripheral devices. The routines may be used by any user-written assembler language programs; most Datapoint products now use these routines whenever possible.

The utility routines are linked into a program at execution time, using DOS Functions 13 and 15. Since DOS Function 15 uses the UTILITY/LNK file, UTILITY/LNK must be available in order to use UTILITY/REL. Member sizes listed below are approximate; use the LIBSYS program or DOS Function 15, subfunction 0 to determine the exact size of a member. For definitions of relocatable code terms such as "PAB", "external definition", and so on, consult the SNAP/3 User's Guide.

46.1 Printer Drivers

The members LOCAL, SERVO, SCREEN and FILE output print lines to the selected device. All of the print drivers have a PAB flag of 'T', meaning that they can be loaded starting at any 256-byte page boundary. If the member is loaded starting at some location other than a page boundary, the load will succeed but the driver will not work. On exit all registers should be considered indeterminate unless their contents are specified below.

LOCAL - Print a line to a local (address 0303) printer. Approximate size 0461 bytes.

SERVO - Print a line to a servo (address 0132) printer. Approximate size 02432 bytes.

SCREEN - Display a line on the console CRT. Approximate size 0352 bytes.

FILE - Print a line to a disk file. Approximate size 02341 bytes.

46.1.1 Print Driver Routines

The print drivers all use the same set of entry points: POPEN$, PRINT$, PCLOSE$, PSUSP$ and PRESTRT$. Each entry point is a "JMP" vector to the appropriate routine within the driver, and the five entry points are the first 15 bytes of each library member. Each of these labels is also an external definition of the library member. All five of the routines simply perform their specified function and return. There are no error conditions on return.

POPEN$ - Open the selected output device. If a printer (LOCAL or SERVO) is the output device, this routine checks for printer ready and for printer allocation (via DOS Function 9). If the printer is already allocated as a resource for the other partition, or if the printer is not ready, a message specifying "PRINTER OFF-LINE" flashes on the screen. Depressing the KEYBOARD key causes the routine to abort via ERROR$. When the printer is available, it is allocated as a resource to the present partition and the routine returns.

If the output device is a disk file, HL must point to an opened LFT entry, and the A register should contain the queueing option byte. The queueing option is either zero for no queueing or non-zero for queueing. If queueing is selected output will be queued to the end of an existing file; with no queueing the output will start at the beginning of the specified file. A disk file created using this routine will have a special user LRN zero written to the disk. This record is a configuration sector similar to that used by EDIT, but storing different information. The information stored allows quickly positioning to the end of the disk file to queue additional output to it.

If the output device is the screen, device open is not necessary.

PRINT$ - Output a line to the selected device. On entry, HL points to a line of characters starting with an ASA control character and terminated by an 015 (EOL). The print line may not contain any control characters or space compression. The routine returns after the line has been output.

PCLOSE$ - Close the output device. If the output device is a printer (LOCAL or SERVO), the routine will form feed if necessary to maintain page parity and will deallocate the printer. If the output device is a disk file, the file will be closed (excess space will be deallocated). If the output device is the console CRT, device close is not necessary.

PSUSP$ - Suspend printing. Call this routine to suspend printing while loading an overlay of your program. Enter with HL pointing to a place to save an LFT entry (16 bytes). The routine will exit with the current output page count in DE.

PRESTRT$ - Restart printing. Call this routine to restart a print file after suspension. Enter with DE containing the page count (saved from PSUSP$), and HL pointing to the saved LFT entry (also from PSUSP$).

46.1.2 ASA Control Characters

The following print control characters are recognized by the print drivers:

space - single space before printing
+ - suppress line feeds
0 - double space before printing
- - triple space before printing
1 - skip to top of form
any other character - treated as space

Whenever a line is sent to the PRINT$ routine, the first character of the line is assumed to be a print control character as defined above and the appropriate action is taken. The print control character is not sent to the output device as an output character.

46.2 SECINOUT Drivers

The drivers SEC5500, SEC2200, SECPS and SECABP are collectively called the SECINOUT drivers because they move 256-byte blocks between processor memory and the disk controller buffer. All the sector in/out drivers are non-page sensitive and may be loaded anywhere in the user's program area.

SEC5500 - Moves 256 bytes to or from the disk buffer on a 1800, 5500, 6000 or 6600 processor, when neither PS nor ARC is active. Approximate size 0211 bytes.

SEC2200 - Moves 256 bytes to or from the disk buffer on a 1100 or 2200 processor. Approximate size 0124 bytes.

SECPS - Same as SEC5500 except that it runs when PS is active. Approximate size 072 bytes.

SECABP - Same as SEC5500 except that it runs when ARC is active. Approximate size 0100 bytes.

46.2.1 SECINOUT Driver Routines

The SECINOUT drivers have only two routine entry points, which have no external definitions. The transfer address of the driver is the sector in routine; the transfer address plus three is the sector out routine. Both routines require the same entry conditions:

B = DOS LFN (extended LFNs may be used)
H = MSB of 256-byte input or output memory buffer (must start on page boundary)

On exit all conditions are indeterminate.

CHAPTER 47. UTILITY/SYS

Most of the DOS commands have been put in an absolute library named "UTILITY/SYS". This has the following advantages:

1. Free up some directory and data space.
2. Makes most of the utility programs available on any disk, since UTILITY/SYS can be on any drive on-line.
3. Assures the user that the most current DOS commands will be used.

Using the librarian utility program (LIBSYS), many user programs can also be added to UTILITY/SYS. A few guidelines for programs that can be members of "UTILITY/SYS":

1. Programs should start at 017000 or higher.
2. Programs that use overlays should use DOS function 13 and 14 to access the library.

If you have placed your own programs into UTILITY/SYS, do not overwrite UTILITY/SYS on a partial gen. Instead, MIN the new UTILITY/SYS using a different file name, then use LIBSYS as follows:

```
MIN
(filename UTILITY/NEW)
LIBSYS UTILITY/SYS
REPLACE UTILITY/NEW
END
KILL UTILITY/NEW
YES
```

To display the members in UTILITY/SYS, enter:

```
CAT *
```

When keyboard commands are entered, the specified command will automatically be located as either a separate disk file or a member of UTILITY/SYS. Normally a separate file name is first checked, then the library member. To reverse the normal precedence put a leading * or : in front of the command name. For example:

```
*CHANGE SCRATCH/TXT;X
     or
```

```
:CHANGE SCRATCH/TXT;X
```

See the chapter on the Command Interpreter for details on selection of a command from the disk directory or from UTILITY/SYS.

CHAPTER 48. SYSTEM DESCRIPTION

48.1 System Philosophy

The objective of DOS is to allow maximum use of the capabilities of a Datapoint disk system with a minimum of effort. The DOS disk structure provides dynamic space allocation and fully random file access capability on all supported disk types. Also provided are an extensive set of utility programs to perform many basic data processing functions. In all system utilities the operator commands are as simple as possible while providing a versatile program capability. Error codes and program messages are mostly presented in English, avoiding complex, incomprehensible messages.

Datapoint DOS is a facilities oriented system. It provides utility programs for general use, and extensive system routines for use in assembler coding. DOS is not a supervisory system; it imposes practically no overhead. The DOS facilities provide a base for Datashare, BASIC, and most other Datapoint languages and systems.

48.2 System Structure

DOS occupies only the lower 8K of memory in the processor. Of this 8K, only the lower 2.8K is necessary for the support of the disks. The first 768 bytes of memory (0 - 01377) contain the object code loader, entry point table, and interrupt handler. Object code may be loaded from 01400 upwards, overlaying much of DOS. If object code is loaded below 01400, the code overstores the loader or entry points and results are unpredictable.

The operating system debug facility and the keyboard and display routines reside between 2.8K and 4K, the cassette driver routines from 4K to 5.4K, and the command interpreter from 5.4K to 8K. It is recommended that user programs start at 017000 (octal).

To achieve its small size in memory, DOS uses disk-resident overlays for the disk file opening, closing, and allocation routines. Most of the system error messages also reside in an overlay, allowing fully descriptive messages without using a prohibitive amount of memory. A set of short utility routines

(DOS Functions) uses a separate overlay area.

The operating system uses a single disk controller with at least one physical disk drive attached. Each "on-line" drive -- a drive containing a disk ready to read -- is assumed to contain a valid DOS disk, which will have all necessary system tables and files present and in correct format. This assumption on the part of the system requires caution on the part of the operator if a disk not fitting this description is mounted. If, for instance, a disk has been mounted to be DOSGENed, the operator must not run any programs that will attempt to use the disk before it has been DOSGENed, or an abort will occur indicating system data failure.

DOS is designed to be run interactively by an operator at the processor console. The operator generally enters commands from the keyboard, which the operating system interprets and executes. During execution, status information needed by the executing program is requested from the operator via CRT messages expecting a keyed response.

A DOS utility program (CHAIN) allows execution of predefined processes automatically in a non-interactive fashion, so no operator attention is required. Other utility programs extend this automatic capability such that the system can be made almost completely operator independent if desired.

CHAPTER 49. SYSTEM STRUCTURE

49.1 Disk Structure

49.1.1 Introduction

Any disk used with DOS is a self-contained information structure. A disk contains up to 256 files, each of which is described in system tables on the disk and which resides completely on the one disk. No system information on a disk references any other disk.

The basic structure of disk storage is the file. Files on Datapoint DOS consist of up to 38,400 sectors, or as many sectors as fit on a logical disk, whichever is smaller. The space occupied by a file is mapped in its Retrieval Information Block (RIB), which is the first sector of the file. The Directory stores the name of each file and provides a pointer to locate the RIB, thus completely defining a file.

Space for files is allocated in clusters, a cluster being the smallest allocatable unit of disk space. In general, each cylinder of a disk is divided into 8 equal clusters. On DOS.C systems a cylinder has only 4 clusters. Thus a cluster consists of 3, 6, or 24 sectors on diskette, cartridge, and mass storage systems respectively. The sectors constituting a cluster are always contiguous and never cross the boundary of a cylinder or head. The Cluster Allocation Table (CAT) and the Lockout CAT maintain a record of clusters in use or unavailable for use and clusters free for use.

The RIB maps the file space in segments; a segment is a set of contiguous clusters. A file then consists of a set of segments located randomly on the disk, each segment being a small block of clusters. Within this space, the file is logically continuous, there being no logical discontinuity at the boundary of a segment.

Each sector within a file carries its own identification. The first byte of a sector contains the Physical File Number (PFN) of the file to which it belongs. The PFN uniquely identifies a file. The second and third bytes contain the Logical Record

Number (LRN) of the sector. The LRN is a count of sectors in the file, starting with 0 at the first sector, and incrementing by one for each successive sector.

All major tables discussed in this section -- the CAT, Lockout CAT, HDI, Directory, and RIB -- are kept in duplicate. The backup copy of each of the tables helps prevent data loss in event of a read/write error to a system sector.

49.1.2 Disk Space Management: CAT and Lockout CAT

The Lockout CAT indicates locked out cylinders -- cylinders which will not be used by the DOS. Cylinders are automatically locked out at DOS generation if they are found bad by the surface verification. Cylinders may be manually specified for lockout during system generation. Cylinder 0 is always locked out for system use. Each byte of the Lockout CAT represents a cylinder: byte 0=cyl 0, byte 1=cyl 1, byte 2=cyl 2, etc. The byte value is 0377 (017 on diskettes) if the cylinder is locked out, and is 000, otherwise.

The CAT indicates available space for the DOS; CAT updates are performed automatically as space allocation or deallocation is performed. As in the Lockout CAT, each byte of the CAT represents a cylinder. Each bit of a byte represents a cluster of the cylinder: bit 7=cluster 0, bit 6=cluster 1, etc. (For diskettes, bits 7-4 are zero, bit 3=cluster 0, bit 2=cluster 1, bit 1=cluster 2, and bit 0=cluster 3). If a bit is set (1), the cluster it represents is either in use by a file or locked out; if a bit is clear (0), the cluster is free.

The CAT and Lockout CAT observe some fixed format rules:

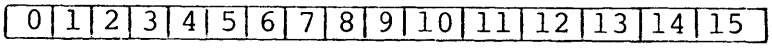
Byte 0 is always 0377
Byte 1 through n may be any value as described above (n is the number of cylinders on the disk)
Bytes n+1 through 0376 are 0377 (except for directory mapping bytes, if used.)
Byte 0377 is any value. This is the auto-execute PFN and is normally zero.

49.1.3 Files: HDI, Directory Mapping Bytes, Directory, RIB

The Hashed Directory Index (HDI) provides access to, and controls allocation of, the Directory. Each byte of the HDI represents a directory entry, offset from the beginning of the index by PFN. Thus, byte 0=PFN 0, byte 1=PFN 1, byte 2=PFN 2, and so on. If the value of the byte is 0377 the directory entry it represents is not in use. When a PFN is in use, a hash code (value 0-0376) generated from the file name is placed in the byte. This value indicates the PFN is in use, and is used to speed directory searching when a file is being loacated by name.

Directory Mapping Bytes are a less sophisticated means of Directory access and control, used in DOS. B version 1 and in diskette operating systems. The mapping bytes are bytes 0357-0376 of the CAT. Each byte represents a directory sector (0-15) and the value in the byte represents the number of entries (0-16) in use in that sector.

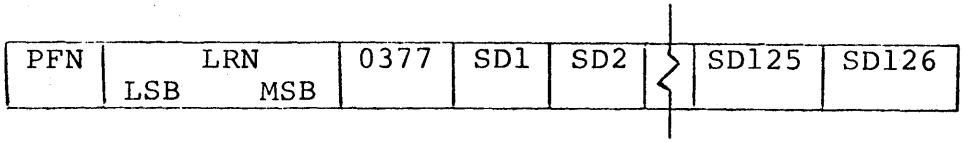
The Directory is 16 sectors (logically referenced as 0-15) containing 256 directory entries, 16 entries per sector. A directory entry contains the name, protection, and subdirectory of a file; it also points to the file's RIB. An empty directory entry is set to all 0377s. Directory entry format:



Bytes 0-1 are the RIB address/protection. (See "Addressing Byte Structures".)
Bytes 2-3 are unused (normally zero)
Bytes 4-11 are the file name. A file name is usually ASCII characters as described in the DISK FILES chapter under File Names, padded with blanks to be eight characters long, but may be any values.

Bytes 12-14 are the file extension. Same format rules as file name.
Byte 15 is the subdirectory number, usually 0377, indicating subdirectory SYSTEM.

A Retrieval Information Block (RIB) maps a file's domain on disk. A file is composed of segments, each segment being composed of contiguous clusters. The RIB contains up to 126 segment descriptors which completely describe the clusters allocated to a file.



Each segment descriptor (SD) is two bytes long (see "Addressing Byte Structures"). A segment descriptor of 0377,0377 indicates the end of the RIB. The fourth byte of a RIB is always 0377. The RIB is always the first sector of the file; the RIB copy is the second sector and is identical to the RIB except that its LRN is 1.

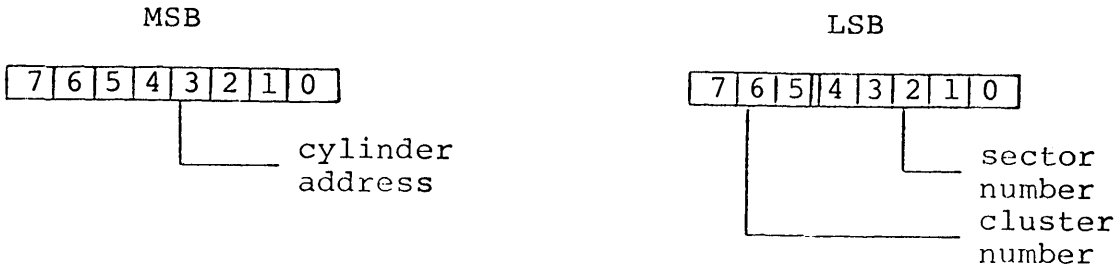
49.1.4 Sector Identification

Every sector of a file contains in its first byte the PFN of the file. The next two bytes are the Logical Record Number (LRN), stored least significant byte first. The PFN and LRN are primarily intended as validation fields when a file record is read. When a file record is written, the PFN and LRN are set correctly; reading a record with a PFN that does not match or an out-of-sequence LRN constitutes a Record Format Error.

Not every sector in the space allocated to a file has this PFN and LRN data. Only sectors that have been used for the file have this information set. Unused sectors may have anything in the first three bytes.

49.1.5 Addressing Byte Structures

49.1.5.1 PDA - Physical Disk Address

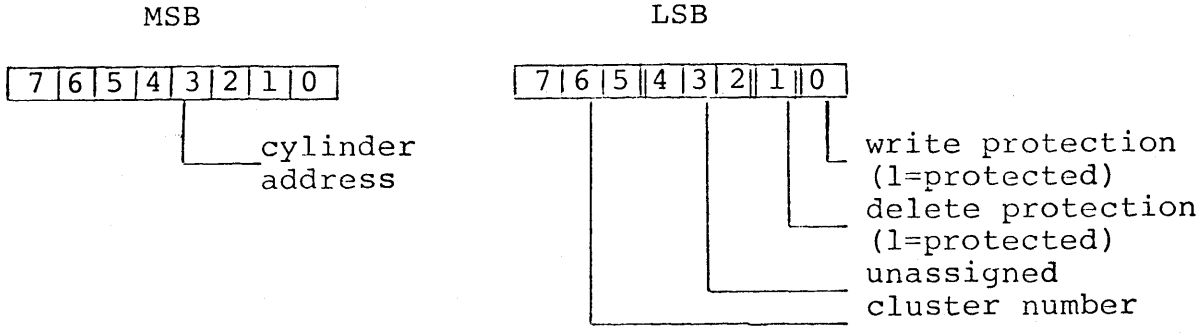


The cluster number references a cluster within a cylinder; values are 0-7 except for diskette systems which use values 0,2,4,6 for clusters 0,1,2,3 respectively. The sector number references a sector within a cluster.

Note: This is the DOS "PDA" and must not be confused with the hardware disk addressing of any particular controller.

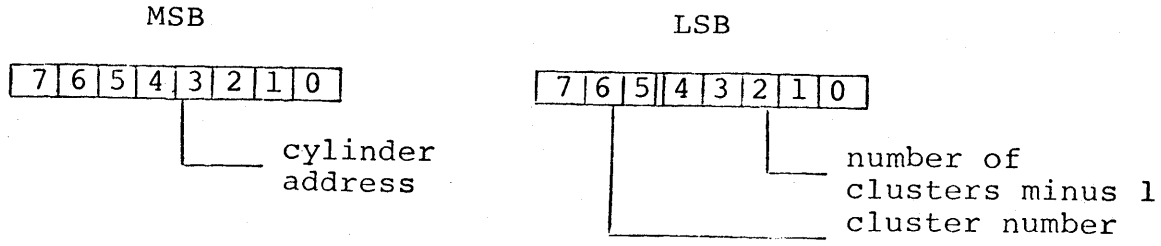
49.1.5.2 RIB Address/Protection

Used in a directory entry to point to beginning of file.



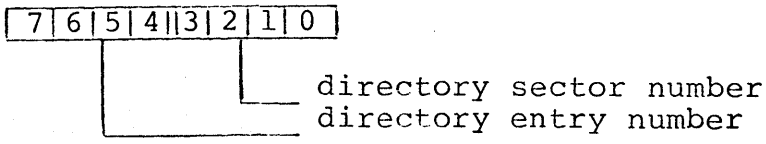
The cylinder address and cluster number, with an assumed sector number of zero, is the PDA of the first sector of the file.

49.1.5.3 Segment Descriptor - used in RIB to define a segment.



The cylinder address and cluster number, with an assumed sector number of zero, is the PDA of the first sector of the segment. The length of the segment in clusters is given by the low-order five bits of the lsb; length can be 1-32 clusters (except DOS. B, 1-10).

49.1.5.4 Physical File Number - used to access directory and HDI



The directory sector number specifies a sector of the directory (0-15). The directory entry number (0-15) specifies an entry within a sector.

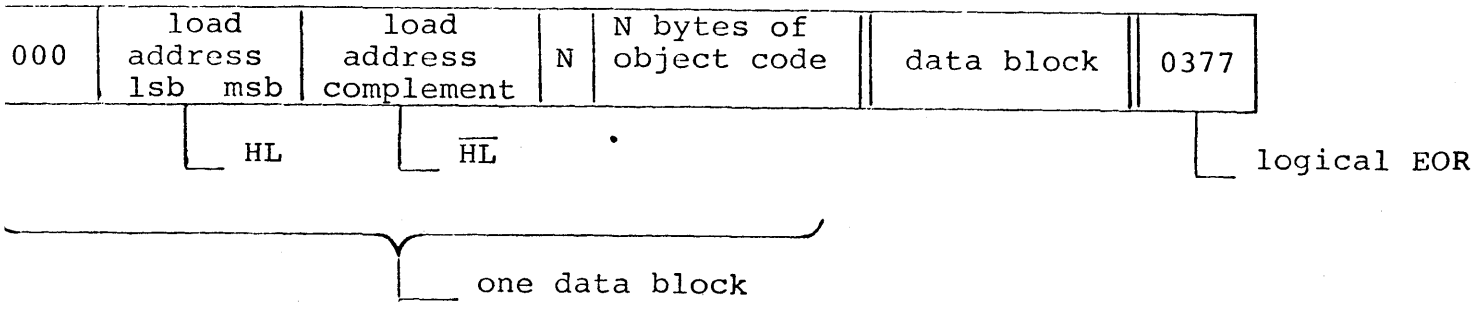
Note: Since directory entries are 020 bytes long, if the low-order four bits of the PFN are set to 0, the resulting value is the byte location of the beginning of the specified directory entry. For example, PFN 0304 references the directory entry beginning at byte 0300 (entry number 014) of sector 4 of the directory.

49.2 Disk Data Formats

The DOS itself does not deal with the user's data below the record level. It only keeps track of where the records are, allowing the user to format the data in any manner he pleases. The user is presented with records that are 253 bytes long, the first three bytes of each sector being reserved for system sector identification as described above. The DOS and its utility programs do make a number of assumptions concerning file structure however, and system operation is much simpler if all files are structured to match these assumptions.

DOS makes assumptions about the structure of text files and about absolute object code files. The structure expected for text files under DOS is described in the chapter on REFORMAT. Any file to be processed by the standard text-handling facilities of DOS must have the standard text format described.

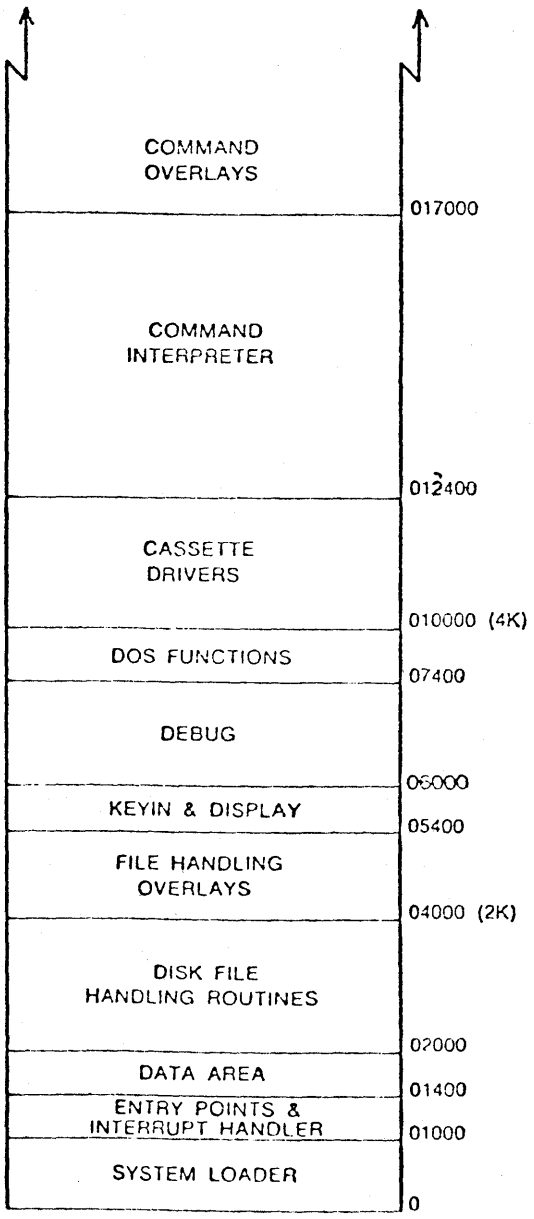
If a file is to be loaded by the system loader, it must be an object code file in the following format:



Note that this is the format of output files from Datapoint assemblers. Any number of data blocks may appear in a record. The leading byte of a data block will always be either 0, indicating a block follows, or 0377, indicating end of record. The special case of N being zero is used to indicate end of file, in which case the HL given is taken to be the starting address of the program loaded.

49.3 Memory Mapping

The DOS occupies memory as shown by the following map:



49.4 Memory Tables

49.4.1 Entry Point Tables

Three entry point tables exist within the DOS. These tables consist of a group of jumps to the various routines made available to the user. These jumps allow the system to be changed without requiring the user to modify his programs. To assure compatability between operating systems and for future versions of DOS, any calls to system routines should use the documented entry points only.

The first entry point table is located between 01000 and 01377. It contains entry points to the routines in the loader (the loader itself, the basic disk read and write drivers, and the interrupt handler) and to the DOS file handling routines. It also contains in-line routines to increment and decrement the HL registers.

The second entry point table is located between 010000 and 010066 and contains entry points to the cassette handling routines.

The third entry point table is located between 013400 and 013452 and contains entry points to routines within the command interpreter. The availability of the command interpreter routines makes small command tasks easy to implement.

See the chapters on System Routines and Routine Entry Points for details on the routine functions and entry point locations.

49.4.2 Logical File Table

The major working table in the system is called the Logical File Table (LFT) and is located from 01544 through 01643. It contains all of the information required by the file handling routines for every file which is currently open (a maximum of three files may be open at any one time - logical files one, two, and three). Once the user has opened a file by its symbolic name, he deals with it by the logical file number under which it was opened. The Logical File Number (LFN) specifies which LFT and which disk buffer memory page are to be used for a file.

The LFT contains for each entry the following information in the order shown (the number in parentheses is the number of bytes used for the item):

PFN	(1) -	Physical File Number, PFN of the file referenced by this LFT
PDN	(1) -	Logical Drive Number (bits 3 - 0) Protection (bit 7 set indicates delete protection, bit 6 set indicates write protection) New Space Allocated flag (bit 5) set if new space has been allocated to this file.
LRN	(2) -	Next Logical Record Number, system LRN of next sequential sector
BLRN	(2) -	Base LRN, first LRN in current segment (system LRN)
CSD	(2) -	Current Segment Descriptor The CSD and BLRN describe the current file segment and allow quick calculation of the PDA to be read/written by treating the LRN as an offset from BLRN. If the desired LRN is not in the current segment, the RIB is re-read and a new current segment established.
RIBCYL	(1) -	Physical Disk Address of RIB, MSB
RIBSEC	(1) -	Physical Disk Address of RIB, LSB Storing the RIB PDA allows quickly locating the RIB when it must be accessed for getting a new segment descriptor, for allocation updates, or for file closing.
MAXLRN	(2) -	Largest system LRN referenced (read, written, or positioned to) since the file was opened. Used for space deallocation at close if new space allocated flag is set.
LRNLIM	(2) -	Largest LRN allowed. Obsolete field, now unused.
BUFADR	(1) -	Current controller buffer byte address, used for byte transfers to or from the disk controller buffer.
XXXXXX	(1) -	Unused

16 bytes total

There are actually four LFT entries (01544-01563, 01564-01603, 01604-01623, 01624-01643) to correspond to LFNs zero through three. The LFN used for a file specifies which buffer page to use for the disk transfer operation. LFN 0 uses buffer page zero (or 4, 8, or 12), LFN 1 uses buffer page one (or 5, 9, or 13), and so on. The larger buffer page numbers are available

on 4K disk controllers and are specified by the high-order bits of the LFN given to the system routine used. Not all routines recognize the page-select feature of LFN, check the description of each routine in System Routines.

Buffer page zero is a special case and is reserved for system use because the DOS needs a buffer into which it can read the RIB if it is necessary to determine a new current segment when a given access is made. This need is only critical on writes, when the buffer contains the information to be written to the disk. On reads, the user's data will always be the last item to be read and and page zero may be used. Always be careful in use of buffer page zero, however, since an access involving a different logical file may cause logical file zero's disk buffer to be loaded with a RIB. Also, the zeroth disk controller buffer is always used by the system loader in transferring data to memory. Page zero is used so that an overlay may be loaded or another program can be chained to without disturbing any of the standard (one through three) logical files. LFN zero has one final peculiarity, CLOSEs have no effect when issued on LFN zero. Neither space deallocation nor updating of the protection field occur when logical file zero is closed.

The DOS loader uses a set of locations in memory between 4 and 022 to perform the functions of an LFT entry during the loading process. It knows, however, that an object file is always sequential and does not have to have the accessing generalization of the main file handling routines. The file handling routines also use these low memory locations for temporary storage of a specified LFT entry to eliminate having to continually index into the LFT. Also, since the basic disk read and write routines use location 5 to indicate which drive is to be used, having the LFT temporarily stored in the low memory locations automatically selects the correct drive for use.

49.5 Disk Overlays

DOS uses disk overlays to reduce its main memory requirements. The overlays are in disk files SYSTEM1/SYS through SYSTEM7/SYS. The memory-resident DOS is stored in the disk file SYSTEM0/SYS. These eight files must reside in PFNs 0 through 7, the PFN corresponding to the number in the file name.

The system overlay files load into memory between 04000 and 05400 and are loaded by the system as needed. The functions of the overlays are:

```
SYSTEM1/SYS - PREP   - create a new file
SYSTEM2/SYS - CLOSE  - close a file
SYSTEM3/SYS - OPEN   - open an existing file
SYSTEM4/SYS - ALLOC  - allocate more space for a file
SYSTEM5/SYS - ABORT  - display an error message
SYSTEM6/SYS - SCREEN - initialize a RAM display screen
```

SYSTEM7/SYS is the DOS Fuction overlay and is described in the DOS Function section of the chapter on System Routines. The DOS Functions are short overlay routines and load into a separate area of memory. Also, the first sector of SYSTEM7/SYS is used to store subdirectory names (see the SUR command).

When DOS needs an overlay file, it searches for the file on the booted drive.

49.6 The Command Interpreter

The command interpreter resides in locations 013400 through 016777. The command interpreter receives command lines from the keyboard, as described in the chapter on Operator Commands, storing the command line in memory in the Monitor Communication Region (MCR$, location 01400 through 01543). When the line is terminated (ENTER key, 015), the stored command line is scanned and the indicated command program is loaded and executed.

While the command interpreter is waiting for character entry from the keyboard, it runs a test on the disk buffer memory. As soon as a character is ready from the keyboard, the disk buffer memory test is terminated and the normal keyin routine is entered. Even just sriking the CANCEL key will terminate the disk buffer memory test. If an error is detected by the disk buffer memory test, the message "DISK BUFFER FAULT" is displayed and the screen is rolled up one line.

When the command interpreter is initially entered via the entry point DOS$ it will execute the program set for auto-execution if there is one. If the KEYBOARD key is depressed, auto-execution is not performed. The AUTOed program will also be run any time the system returns to DOS$; exit routines EXIT$ and ERROR$ return via this entry point.

When a command line has been entered, the command interpreter

must attempt to locate and load the specified command program. If the command is obviously bad (a null entry line) the interpreter immediately displays "WHAT?" and waits for a new line. A pound sign (#) for invoking DEBUG is also treated as a special case, causing the interpreter to immediately go to DEBUG. Normally the first field on the command line will be normalized to the form shown below and the file thus specified will be searched for. The sequence of searching for a requested program depends on the format of the command line.

If the operator entered a leading "*" or ":" as part of the command name, a flag called UTILSW (UTILity SWitch) is set, indicating that the specified command is to be located as a member of UTILITY/SYS. If a drive specification was entered as part of the command name, the search goes only to the specified drive, as indicated in the sequence shown below.

The first test the interpreter performs is to check the drive specification entered, if any. If the drive specification is invalid, an error message is displayed and a new command line requested. If the drive specified is valid, or if no drive was specified, the interpreter searches for the command as outlined below.

1. If a drive was specified:
   a. If UTILSW is set:
      (1) Open UTILITY/SYS on the specified drive. If the file is missing or if the specified command is not a member, say "WHAT?", else run the program.
   b. If UTILSW is not set:
      (1) Attempt to open the command file on the specified drive. If successful, run the program. Else:
      (2) If no extension was specified in the command name, open UTILITY/SYS on the specified drive and search for the command as a member of the library. Else:
      (3) "WHAT?" and get another command.
2. If no drive was specified:
   a. If UTILSW is set:
      (1) Open UTILITY/SYS on the booted drive and search for the command as a member of the library. Else:
      (2) Try to open command as a file on booted drive. Else:
      (3) Check for command in UTILITY/SYS on any drive. Else:
      (4) Try to open command as a file on any drive. Else:
      (5) "WHAT?" and get another command.

b. If UTILSW is not set:
   (1) Try to open command as a file on booted drive. Else:
   (2) If no extension was specified in the command name, open UTILITY/SYS on the booted drive and check for the command as a member of the library. Else:
   (3) Try to open command as a file on any drive. Else:
   (4) If no extension was specified in the command name, open UTILITY/SYS on any drive and check for the command as a member of the library. Else:
   (5) "WHAT?" and get another command.

The command interpreter uses lexical scanning routines to interpret the entered command line. These routines are available for user programs and are described in the chapter on System Routines. The command interpreter scans up to four file specifications from the command line. The file name scan is terminated by a semicolon (;) or end-of-string (015). The file specifications are entered in a normalized symbolic form into the corresponding logical file table entries (0 through 3). The normalized form is not the same as normal LFT information, the LFT area simply provides convenient storage for the file specifications. If desired (and it usually is), the open routine can open a file using the LFT in which the file name used for the open is stored. The format of the normalized form is shown here:

DRCODE	(1)	- Drive select code: logical drive number in binary, no drive spec (0377), invalid drive spec (0376)
0377	(1)	- PDN location of normal LFT, set to 0377 to indicate the LFT is closed.
FILENAME	(8)	- File name specified, padded with trailing spaces to 8 characters. Eight spaces if no name given.
FILEEXT	(3)	- File extension specified. Three spaces if no extension entered.
DRSPEC	(3)	- Logical drive specification (spaces if no spec).

When a program receives control from the command interpreter, LFTs one through three (zero was used to load the program itself) contain normalized entries as indicated above, and MCR$ still contains the command as entered, so the program can retrieve information from its command line. If a program is auto-executed, none of this command line information is available, so any program which tests for information as provided above can not be auto-executed. Conversely, any program intended for

auto-execution must not look for command information. The command AUTOKEY is provided to allow automatic execution of programs requiring command line information.

CHAPTER 50. INTERRUPT HANDLING

50.1 Interrupt Mechanism

Datapoint 1100, 1800, 2200, 5500 and 6600 processors feature a one-millisecond timed interrupt. Every millisecond, a flip-flop indicating "interrupt pending" is set; the setting of this flip-flop occurs independently of processor instruction cycling. At the beginning of an instruction fetch cycle the status of the interrupt pending flip-flop is checked. If the flip-flop is set, and interrupts are enabled, a CALL to the interrupt vector location occurs and the flip-flop is cleared. On 1100 and 2200 processors, the interrupt vector location is address 0. On 1800, 5500 and 6600 processors the interrupt vector location is address 0167444, which normally performs a jump to location 0. While interrupts are active, location zero is a jump to the interrupt scheduler.

The execution of the CALL ends hardware control of the interrupt. Any interrupt service performed, task scheduling, or prioritizing is under software control.

The machine instruction DI (Disable Interrupts) prohibits recognition of the interrupt pending flip-flop, thereby preventing any interrupt calls until recognition of the flip-flop is reactivated by an EI (Enable Interrupts) instruction.

Datapoint processors do not have a structured multi-level hardware interrupt vector mechanism available to the system programmer. Any I/O performed by the system must be monitored by the processor through the use of status flags. Such monitoring is usually performed by a foreground process, typically one process per I/O device.

50.2 Interrupt Scheduler

DOS provides an interrupt scheduler loaded as part of the system boot operation. The scheduler resides between 01201 and 01376 and remains memory-resident. Normal system operation never overstores this scheduler. The basic coding of the scheduler is shown below. The code shown is intended as an example of the structure of the scheduler and is not the exact code used within

DOS.

```
INTRPT    DI                  Disable interrupts
          BETA                Use BETA mode
INT0      CALL   RETURN       Perform each of processes 0
INT1      CALL   RETURN           through
INT2      CALL   RETURN             three
INT3      CALL   RETURN
          MLA    *INTSCN      Rotate to the next
          AD     6                one of processes
          LMA                     4 through 7
          AD     INT4-6
          LLA                 HL => CALL address
          PUSH                Jump to the
RETURN    RET                     next CALL
.
INTSCN    DC     0            Rotation counter storage
.
INT4      CALL   RETURN       CALLs for the rotating
          JMP    INTRET           process slots
INT5      CALL   RETURN
          JMP    INTRET
INT6      CALL   RETURN
          JMP    INTRET
INT7      CALL   RETURN
          XRA                 Reset the scan pointer
          MSA    *INTSCN          after calling process 7
INTRET    ALPHA               back to ALPHA mode
          EI                  Enable interrupts
          RET                 Back to the background
```

All processing performed on an interrupt call is called "foreground", the processing interrupted is referred to as "background". Foreground processing begins with the DI instruction labeled INTRPT above and ends with the RET instruction terminating the scheduler. The above scheduler illustrates the fundamental rules of foreground code on a Datapoint processor:

1. Interrupts must be disabled during foreground processing. The scheduler disables interrupts initially and does not enable interrupts until immediately before terminating. Foreground processes must not enable interrupts or invoke any DOS routines which internally enable interrupts.

2. Foreground processing is performed in BETA mode, background processing in ALPHA mode. The scheduler sets the machine modes; foreground processes should not change the mode.

3. Foreground processes are CALLed routines and must return with the stack in the same condition as on entry. Each CALL instruction INT0 through INT7 can be used to call a foreground routine. The scheduler itself uses a simple RET to return to background processing; if any foreground routine modifies the stack, the scheduler could exit to the wrong location. Even if the scheduler manages to return properly, the background process uses the same stack (there is only one stack) and any modification performed by foreground routines could be fatal to the background processing.

4. Register contents on entry to foreground processing may be undefined. Normally the BETA mode registers and condition flags will be the same on entry to foreground as they were at the conclusion of foreground processing on the previous interrupt, so contents on entry can be considered as known. Under PS, however, the BETA mode registers and condition flags are not preserved, since they are used by PS and by the other partition. Even when PS is not in use, register contents cannot be predicted if there is a possibility of multiple foreground routines being active. If a single routine is active, registers may be preserved; if another routine or two is made active, they may modify the registers used by the first routine, effectively destroying any expected contents.

50.3 Active Processes

Each of the labeled CALLs in the scheduler, INT0 - INT7, is called a foreground "process" or, sometimes, an interrupt "slot", and is referenced by number 0 through 7.

Normally each foreground process is inactive, since each CALL invokes only a RETURN to the scheduler. A process is made active by overstoring the address RETURN following the process CALL with the entry address of a desired foreground routine. The address so stored is called the "state" of the foreground process. (Two DOS routines, SETI$ and CS$, set the state of a process.) Thus if the address PRINT is stored LSB, MSB in INT1+1 and INT1+2 (the address area following the CALL at INT1) the state of foreground process 1 would be PRINT. Once a process has been made active -- given a state -- it can again be made inactive by storing the adress RETURN back into the two bytes following the CALL. (Two DOS routines, CLRI$ and TP$, terminate processes in this manner.) While a process is active, the routine it calls will be performed once every interrupt cycle, or every fourth cycle depending on the

slot number used.

The scheduler is structured to provide four "one-millisecond" processes and four "four-millisecond" processes. The "one-millisecond" and "four-millisecond" designations refer to the length of time between sequential executions of the process. Interrupt slots 0-3 are one-millisecond processes; each process is executed every time an interrupt occurs. Interrupt slots 4-7 are four-millisecond processes; one of the four is executed every time an interrupt occurs, so any one process is executed only every fourth millisecond.

50.4 Timing Considerations

The most severe constraint on foreground routines is timing, mainly the total length of time required to execute. Since an interrupt occurs every millisecond, the total amount of time spent in foreground must be less than 1000 microseconds. Thus the amount of time spent executing each active foreground routine and the interrupt scheduler itself (130 microseconds on a 2200) must total less than a millisecond. If the time spent in foreground is more than a millisecond, the interrupt pending flag will already be set when the interrupt scheduler executes its final RET, so an interrupt call will immediately occur and no background processing will be performed.

Also, if more than one millisecond is spent in foreground, interrupts can be dropped. The interrupt pending flag has only "off" and "on" values. If an interrupt signal occurs while the flag is already on, it simply stays on and the occurrence of the interrupt pulse has no effect -- the interrupt is lost. If, for example, 1200 microseconds is being spent in foreground on each interrupt, only 5 interrupt calls will occur in a 6 millisecond time interval. One interrupt will be lost because the flag was already set when its signal occurred. In a similar fashion, interrupts can be lost if interrupts are disabled for too long in background.

Another timing concern is "jitter". Jitter describes the variation in interrupt timing: it is not exactly one millisecond between interrupt calls. The timing variation occurs mainly because of time spent in background with interrupts disabled. If background processing disables interrupts for 200 microseconds (200 microseconds of jitter) it could be 1200 microseconds between interrupt calls if the interrupt pending flag were set immediately after interrupts became disabled. An additional source of jitter is time spent in foreground processes. Any variation in the

execution time of process 0 appears as jitter to process 1.

Jitter must be taken into account when designing any program structure. If an external device is being serviced by interrupts and the device presents a character for input every 1.4 milliseconds, jitter must not exceed 400 microseconds. If the jitter were over 400 microseconds, a character could appear ready and then be overstored by the next character before an interrupt occurred to service the device. A good guideline is 200 microseconds maximum for any foreground process.

50.5 DOS Interrupt Routines

DOS provides four utility routines for interrupt processing. Use of these routines simplifies interrupt process coding and helps assure DOS compatibility. For full descriptions of parameterization of these routines, see the chapter on System Routines.

50.5.1 SETI$

SETI$ changes the state of a foreground process. SETI$ is usable only from background and is generally used to initiate a previously inactive process. The routine accepts a specified address and stores the address following the CALL instruction of a specified interrupt slot. Even if the process was previously active, the new state is stored over the old state.

50.5.2 CLRI$

CLRI$ terminates a foreground process. The address of RETURN (see sample scheduler above) is stored following the CALL of the specified process number. Any routine active from that interrupt slot is then inactive. CLRI$ is used from background.

50.5.3 CS$

CS$, like SETI$, changes the state of a foreground process, but is used from foreground. A call to CS$ affects only the process performing the call. CS$ changes the state of the process to the address of the instruction following the "CALL CS$" and returns -- not to the invoking routine -- but to the interrupt scheduler. Due to the stack manipulations performed by CS$ it must be called only from the outermost stack level of a foreground

routine; it must not be called from a routine called by the main routine. CS$ does not enable interrupts.

50.5.4 TP$

TP$, like CLRI$, terminates a foreground process and is itself called from foreground. TP$ affects only the foreground process from which it is called, setting the state of that process to RETURN to deactivate the process, and returning to the scheduler. Like CS$, TP$ must be called only from the outermost stack level of a foreground routine. TP$ does not enable interrupts.

50.6 Programming Considerations

50.6.1 Background Code

If interrupt processing is to be used, the mainline program code must be written "interruptable" with the realization it may be interrupted anytime interrupts are not disabled. For most processing, no particular concern is necessary, since if the interrupt processes are coded correctly the stack, registers, and condition flags are unchanged after the interrupt process; the background code will never notice the interruption. Coding for I/O device handling is the most critical part of interruptable code, since during interrupt processing the selected I/O device can change.

Interrupts must be disabled any time the currently selected I/O device status is critical: between addressing the device and testing status, between addressing the device and issuing a command, and so on. At the same time, interrupts must not be disabled for too long a time, due to introducing excessive jitter or even dropping interrupts. It is especially important to be certain interrupts are enabled for at least one instruction cycle in any wait loop lest interrupts be delayed for the duration of the loop.

If the background code uses BETA mode, interrupts must be disabled all the time BETA mode is in use. If an interrupt occurs while in BETA mode, the registers and condition codes will be modified by the scheduler and foreground routines and results to the background program could be disastrous. Background code should not generally use BETA mode.

All DOS utility routines are completely interruptable and disable interrupts for a maximum of 200 microseconds. DOS routines generally return with interrupts enabled.

50.6.2 Foreground Code

Duration of foreground routines is of primary concern. If a routine is too long to execute in a single interrupt cycle, its operation must be split using CS$ or successive four-millisecond processes. Foreground routines should never use a wait loop; they should instead return, using the delay of background processing to wait for the next interrupt.

Additionally, foreground routines:

1. Must not enable interrupts.
2. Must exit with the stack in the same condition as on entry.
3. Must not use mode instructions.
4. Should not assume register conditions have been preserved.

Be sure to terminate foreground processes when they are no longer needed. A process left active uses up machine time. When a program finishes, any active foreground processes remain active upon return to DOS. These foreground processes at best slow down the system, and may cause CALLs to locations that have been overstored by later programs, causing unpredictable results.

DOS itself uses foreground processing in only a few instances: the cassette driver routines, the DEBUG P-counter display, the delay function (DOSFUNC 8), and the relocatable servo printer driver.

CHAPTER 51. SYSTEM ROUTINES

51.1 Parameterization

Parameters are passed to the subroutines through the registers. In the discussion of these parameters, the following abbreviations will be used:

LFN - Logical File Number times 16
LRN - Logical Record Number (the user' s LRN)
PFN - Physical File Number
LFT - Logical File Table

The following definitions apply to the descriptions of the system routines.

drive number - indicates a logical drive number (0 through N, where N is the maximum number of logical drives supported by the DOS in use). In some routines, 0377 is used to indicate that all drives are to be checked.

name - the address of a field containing exactly eleven bytes. The first eight bytes are the file name and the last three bytes are the file extension by command interpreter convention. The name characters may be any eight bit combinations except the first character must not be a 0377. The command interpreter requires that all characters be letters or digits.

=> - indicates an address pointer. This symbol will be used when describing routine entry or exit conditions to indicate that a specific register pair contains an address which is the memory location containing the specified information. Thus "HL => string" means the HL register pair contains the memory address of the first character of a string.

extended LFN - LFN referencing a physical disk buffer. The normal DOS LFN values are 0, 16, 32, and 48 (0<4, 1<4, 2<4, and 3<4), and specify usage of a particular LFT and of a particular page of disk buffer memory (pages 0 through 3). Some of the system routines for DOS.D, DOS.E, and DOS.G allow specification of larger LFN

values, allowing use of buffer pages 4 through 15. The extended LFN is specified as buffer page number times 16. The LFT used by a specified LFN is determined by the low-order two bits of the buffer page number. Thus, pages 0, 4, 8, and 12 use LFT 0, pages 1, 5, 9, and 13 use LFT 1, and so on.

51.2 Exit Conditions

If a routine fails to perform as expected, some indication must be made that the expected action did not occur. This indication is given by the condition flags in the processor being set in a special manner or by control being transferred to a trap location instead of returned via the subroutine mechanism. The 'Exit conditions' section of each subroutine description shows the register contents and condition flags of interest when the routine returns.

51.3 Error Handling

Minor errors are indicated by the Exit Condition of the routine called. Major errors cause a trap -- an automatic seizure of program control by the operating system. The trap for each type of error transfers control to a specified location, which will display an appropriate error message.

Minor errors are always non-fatal; the program can test the Exit Conditions and determine what action to take. Major errors can be fatal or non-fatal. When a system trap occurs, the system will simply display a message and restore itself, causing a fatal program error. Many major error traps can be intercepted by the program and given special treatment, as described in the section on TRAP$ below.

51.4 Foreground Routines

The chapter on Interrupt Handling contains a complete discussion on the functioning and use of the foreground handling and should be consulted for an understanding of the following routines.

51.4.1 CS$ - Change Process State

CS$ changes a foreground routine's state. It is called by the executing foreground routine and causes its execution address to be changed to the address following the CALL CS$. Execution will not continue at the new address until the next interrupt occurs. CS$ is normally called from the outermost stack level (level 0) of an active foreground process. Calls to CS$ from deeper stack levels of the routine must be very carefully planned and are not recommended.

Entry point: 01033

Parameters: on subroutine stack - see the chapter on Interupt Handling

Exit conditions: return is made to the scheduler

51.4.2 TP$ - Terminate Process

TP$ deactivates the process called by storing the address of a return instruction in the process call. TP$ is entered by a jump instruction, not a call. TP$ is invoked from the outermost stack level (level 0) of an active foreground process.

Entry point: 01036

Parameters: on the stack - see the chapter on Interrupt Handling

Exit conditions: no exit, returns to Interrupt Scheduler

51.4.3 SETI$ - Initiate Foreground Process

SETI$ activates the interrupt process specified by the parameter in the C register (0-7) by storing the address given in the D and E registers into the call instruction for that process and enables the interrupt handler. Interrupt processes zero through three are executed every millisecond while four through seven are executed every fourth millisecond.

Entry point: 01041

Parameters: C = process number (0-7)
DE = address of foreground process

Exit conditions: B,D,E unchanged
H,L = 0

51.4.4 CLRI$ - Terminate Foreground Process

CLRI$ deactivates a foreground process by storing the address of a return instruction into the process call specified by the parameter in the C register (0-7) and enables the interrupt handler.

Entry point: 01044

Parameters: C = process number (0-7)

Exit conditions: B unchanged
H,L = 0

51.5 Loader Routines

There are two levels of disk handling routines. This section describes the lower level routines which reside in the loader and require numbers physically describing the drive, cylinder, sector, buffer, and file. The section on File Handling Routines describes the upper level routines.

INCHL and DECHL are described in this section only because they are used by the DOS at all levels and because these two routines are loaded as part of the bootblock. In general, the other routines described in this section are not used by typical user programs; most user programs will be better served by the higher level routines described in the section on File Handling Routines.

51.5.1 BOOT$ - Reload the Operating System

BOOT$ loads and executes the operating system (PFN 0 on the booted drive). This action does not affect the interrupt handling facility between 01000 and 01377. Since BOOT$ requires that the operating system always be loaded specifically from the booted drive, BOOT$ should normally only be used in cases where EXIT$ is unusable, such as when the disk handling routines have been overstored. BOOT$ does not close any files before reloading the DOS.

Entry point: 01000

Parameters: none

Exit conditions: does not return

51.5.2 RUNX$ - Load and Run a File by Number

RUNX$ loads the physical file specified and begins its execution. If the file cannot be loaded, a jump to BOOT$ occurs.

Entry point: 01003

Parameters: A = PFN
C = drive number

Exit conditions: does not return

51.5.3 LOADX$ - Load a File by Number

LOADX$ loads the physical file specified and returns with the starting address in HL if the load was successful.

Entry point: 01006

Parameters: A = PFN
C = Drive Number

Exit conditions: Carry false: HL = Starting address of file
Carry true: A=0 if file does not exist
1 if drive off-line
2 if directory parity fault
3 if RIB parity fault
4 if file parity fault
5 if off end of physical file
6 if record of illegal format

51.5.4 INCHL - Increment the H and L Registers

INCHL increments the sixteen bit value in the HL registers by one. If the routine is entered at INCHL+2, the sixteen bit value in the HL registers will be incremented by the number in the A register.

Entry point: 01011 (01013 for increment by A)

Parameters: HL = number to be incremented

A = increment value if INCHL+2 used

Exit conditions: HL incremented
A equal to the H-register
B,C,D,E unchanged
all condition flags undefined

51.5.5 DECHL - Decrement the H and L Registers

DECHL decrements the sixteen bit value in the HL registers by one. If the routine is entered at DECHL+2, the sixteen bit value in the HL registers will be decremented by negative the number in the A register (e.g., for decrement by 2, A is set to -2).

Entry point: 01022 (01024 for decrement by -A)

Parameters: HL = number to be decremented
A = decrement value if DECHL+2 used

Exit conditions: HL decremented
A equal to the H-register
B,C,D,E unchanged
all condition flags undefined

51.5.6 GETNCH - Get the Next Disk Buffer Byte

GETNCH gets the character from the physical disk buffer location pointed to by memory location DOSPTR (location 026) from the disk buffer currently selected, and then increments the contents of location DOSPTR.

Entry point: 01047

Parameters: DOSPTR = disk buffer address (0-255)

Exit conditions: A = character from disk buffer
(DOSPTR) = (DOSPTR)+1
B,C,D,E,H,L all unchanged

51.5.7 DR$ - Read a Sector into the Disk Buffer

DR$ causes a sector to be transferred from the disk to one of the disk controller buffers. The drive number is given in the least significant bits (the others are ignored) of location TFT+PDN (location 5). (The number of bits ignored depends upon the particular DOS in use). The physical disk address LSB is given in the E register and the physical disk address MSB is given in the D register. The disk controller buffer number times sixteen is given in the B register. Interrupts are disabled by this routine for a maximum of 100 microseconds.

Compatibility note! The physical disk address format will vary among different DOS; the user's program should not make assumptions regarding this format if the program is to be transportable between different DOS. The most significant byte (MSB) is generally a cylinder number, and the least significant byte (LSB) is a sector address within a cylinder. The least significant byte especially will vary among DOS. In general, the only safe way to insure a valid, proper physical disk address (PDA) is to get it as a returned item from a system routine (POSIT$ or one of the DOS FUNCTIONs described later). User program generation or manipulation of physical disk addresses is strongly discouraged.

If parity faults are detected, DR$ retries the read operation four to ten times (depending on the type of disk drive in use) before returning with an abnormal exit status. This routine is used by all higher-level DOS disk routines, so the same retry count applies to all DOS disk operations.

Entry point: 01052

Parameter:
B = extended LFN
D = physical disk address MSB
E = physical disk address LSB
TFT+PDN (at loc 5) = logical drive number

Exit conditions:
B,D,E,TFT & PDN all unchanged
Carry false if read successful
Carry true and Zero false if drive off-line
Carry true and Zero true if parity fault

51.5.8 DW$ - Write a Sector from the Disk Buffer

DW$ causes the contents of one of the disk controller buffers to be transferred to a sector on the disk. If the physical write protection on the specified drive is enabled, DW$ will beep continuously until the protection is disabled.

There are two types of write protection in the disk operating system. The first type is a physical protection that is part of the disk drive hardware, which will cause DW$ to beep if set. The second type of write protection is a logical protection that is connected with each file on a disk. A bit exists in the directory entry for each file which, if set, will prevent the higher-level routines (for example, WRITE$) from calling the DW$ routine. It is important not to confuse these two types of write protection. All references to write protection that follow refer to the logical protection on each file and not to the physical protection of the drive itself.

In DOS, DW$ uses the write/verify mode of the disk controller. Thus all writes performed by the DOS use this mode of writing. As in the DR$ routine, several retries will be made if parity faults are detected before abnormal exit will occur. In all other respects, DW$ is similar to DR$.

Entry point: 01055

Parameters:
B = extended LFN
D = physical disk address MSB
E = physical disk address LSB
TFT+PDN (at loc 5) = logical drive number

Exit conditions:
B,D,E,TFT & PDN unchanged
Carry false if write successful
Carry true and Zero false if drive off-line
Carry true and Zero true if parity fault

51.5.9 DSKWAT - Wait for Disk Ready

DSKWAT waits for disk ready, controller ready, no disk I/O transfer in progress, and drive online to all be true. If the drive is not online, return is made with the carry flag true, the zero flag false, and interrupts enabled. Otherwise, exit is made with interrupts disabled. This routine is obsolete and is not available under some systems (PS, for example). It is recommended that this routine not be used.

Entry point: 01060

Parameters: none (drive checked is the selected drive)

Exit conditions: explained above
B,C,D,E,H,L unchanged

51.6 File Handling Routines

A file is dealt with as a logically contiguous and randomly accessible space. A file is specified by its symbolic name or by its PFN. A LRN within a file is specified by a two-byte number kept within the system (LRN in the LFT). When a file is opened, the LRN is set to two.

There is a distinction between system LRN and user LRN. A LRN in the LFT is a system LRN. System LRN zero is the primary RIB for a file and system LRN one is the RIB backup. System LRN two is user LRN zero (user data sector 0). Logical record numbers supplied to system routines are usually user LRNs. These numbers are converted to system LRNs before being used by the DOS or placed into the LFT. In the routine descriptions below, "LRN" refers to a user LRN unless otherwise specified.

After each record access (READ$ or WRITE$), the LRN in the LFT is incremented. Thus, for sequential accesses, the user need not actually specify which record he is dealing with. However, a routine named POSIT$ allows the LRN to be changed to any value between zero and the upper limit of the file, providing a random access facility. (This upper limit depends upon the DOS in use).

All of the logical file handling routines automatically create or verify the PFN and LRN of the file sector being handled (see Disk Structure).

It must be noted that READ$ and WRITE$ provide sequential processing of file sectors, but do not automatically handle the Datapoint sequential text file format. All necessary end-of-record (015) and end-of-sector (003) bytes must be placed in the disk buffer under program control; the system routines do not provide these control bytes. Likewise, the CLOSE$ routine does not provide any end-of-file mark. To provide a valid text EOF, the user program must write an EOF byte by byte. For Datapoint file formats, see the appendix on Disk Data Formats.

51.6.1 PREP$ - Open or Create a File

PREP$ searches the directory or directories specified for the given name. If the name is found, the file is simply opened for use as the specified logical file number. Otherwise, a new file having the name specified will be created. If a new file is created, an end-of-file by GEDIT convention (six zeros followed by an 003) is written in LRN zero. Whether the file is simply opened or is created, the information describing it is stored in the LFT entry specified so that all subsequent references to that file by its LFN will be able to deal with the correct locations on the disk. If the LFT entry specified is already in use when PREP$ is called, the file that the entry specifies will be closed (see the section on CLOSE$) and the new file opened in its place.

DE is the address of an 11-byte string which is the name of the file being specified (as explained before under the section on Parameterization).

Entry point: 01063

Parameters:
B = LFN
C = drive number or 0377
DE=> file name string

Exit conditions: B = LFN; other registers indeterminate

Traps:

SPACE	A new file must be created and no space is left or no more directory entries are available.
OFF-LINE	The drive specified is off-line.

51.6.2 OPEN$ - Open an Existing File

OPEN$ is similar to PREP$ except for the action taken if the file specified does not exist. In this case, return is made with the Carry condition true (return is made with Carry false if the file exists). In addition, a file may be opened by PFN by setting the D register to zero and setting the E register with the PFN. Action taken to open by PFN is the same as that taken if a name is specified.

Entry point: 01066

Parameters:
B = LFN
C = drive number or 0377
DE => file name string or

D = 0,E = PFN

Exit conditions: B = LFN; other registers indeterminate
Carry true if the file is non-existent

Traps: none

51.6.3 LOAD$ - Load a File

LOAD$ opens the specified file as logical file zero and then calls the system loader to load it into memory. Exit is made with the Carry condition set if the file is non-existent, or if the drive specified (if any) is off line. If the load is successful, return is made with the starting address in the H and L registers.

Entry point: 01071

Parameters: same as for OPEN$ (except B not required)

Exit conditions: B = LFN (always zero)
HL = starting address if good load
Carry true if file non-existent or drive off-line

Traps:

OFFLIN	Drive went off-line after loading began.
RPARIT	File contains parity fault.
RANGE	Loader ran off end of file.
FORMAT	Record of bad loader format found.

51.6.4 RUN$ - Load and Run a File

RUN$ opens the specified file as logical file zero and then calls the system loader to load it into memory. Return is made to the instruction following the call if the name specified cannot be found in the directory or directories specified. If any loading errors occur, the operating system is reloaded. Otherwise, control is transferred to the starting address given by the loader.

Entry point: 01074

Parameters: same as for OPEN$
(except that B is not required)

Exit conditions: If successful, routine does not return;
program from specified file is executed.

Returns if name not found.
DOS reloaded if bad object program load.

Traps: none

51.6.5 CLOSE$ - Close a File

When new space is allocated for a file, a large contiguous piece (up to one full segment) is taken in an effort to keep the file as physically contiguous as possible. When this allocation takes place, a flag in the LFT, called the new space allocated flag, is set. The LFT also contains a number which is the largest LRN referenced while the file was open. When CLOSE$ is called, the file is physically truncated after the largest LRN referenced, if the new space allocated flag is set. Thus, if only a few records of the new space allocated have been used, the rest of the space is freed for use in other files. However, if all of the space is used, the file will consist of a large amount of physically contiguous space. If CHOP$ was called with the D register set to -1 (0377), and the LRN in the LFT has not been changed, a call to CLOSE$ will delete the entire file and remove its entry from the directory.

After the file has been truncated, if necessary, CLOSE$ then writes the copies of the protection bits and old file length limit field that are in LFT entry back into the directory. Therefore, one only needs to change these entries in the LFT and then close the file to have them changed in the directory. This action is the basis for the functioning of the CHOP$ and PROTE$ routines. Since the protection bits and old file length limit field are not changed on the disk until the CLOSE$ routine is called, if one changes these numbers and then, for some reason, reloads the system without calling the CLOSE$ routine (by rebooting the system before the file is closed, for example) the disk will retain the old values.

If deallocation of file space (CHOP$) is to be used following a protection change (PROTE$), the file must be CLOSEd and re-OPENed.

```
DE      NAME                FILE NAME
LC      -1                  DRIVE
CALL    OPEN$               OPEN THE FILE
LC      1                   CHANGE PROTECTION
CALL    PROTE$              CHANGE PROTECTION
```

```
            CALL    CLOSE$              NOW, SET THE PROTECTION
            DE      NAME                RE-OPEN THE FILE
            LC      -1
            CALL    OPEN$
            DE      -1                  CHOP THE FILE
            CALL    CHOP$
            CALL    CLOSE$              AND DELETE IT.
.
.
.
NAME        DC      'SCRATCH TXT'       FILE NAME TO BE DELETED
.
.
.
```

After the protection and file length limit have been stored in the directory, CLOSE$ then vacates the LFT entry specified by storing an 0377 in the second byte of the entry (this is the drive number and 0377 denotes that the LFT entry is not in use). CLOSE$ simply returns if the LFT entry is not in use.

Entry point: 01077

Parameters: B = LFN (16,32,48; 0 => NOP)

Exit conditions: B = LFN; other registers indeterminate

Traps: none

51.6.6 CHOP$ - Delete Space in a File

CHOP$ sets the LFT entry to deallocate file space following the given LRN. If the CLOSE$ routine is called after the call to CHOP$ without the LRN being changed, the space after the specified LRN will be physically deleted from the file, making it free again for allocation by the system. Note that if the D register is set to -1 (0377) upon entry to CHOP$, calling the CLOSE$ routine will completely delete the file from the system (removing its entry from the directory as well as freeing all of its space). When an entry is deleted from the directory, all sixteen bytes of the directory for that entry are set to 0377 (value set by the system generation program for unused directory entries).

Remember that calling CHOP$ only affects the LFT entry and that no physical change on the file is effected until CLOSE$ is called.

Entry point: 01102

Parameters:
B = LFN
DE = LRN if D not 0377
D = -1 (0377) to delete entire file

Exit conditions: B = LFN; other registers indeterminate

Traps:

RANGE	DE was not less than MAXLRN.
DVIOLA	Delete protection is set.
WVIOLA	Write protection is set.

51.6.7 PROTE$ - Change the Protection on a File

PROTE$ changes the file protection bit and/or upper file length limit copies that are kept in the LFT. The protection bits, given in the C register, are changed only if the least significant bit of the C register is a one. The old upper file length limit field is changed only if the sign bit of D is one on entry. Therefore, setting the number to zero prevents the limit field from being changed. The file length field is obsolete and is no longer used by the DOS; it is maintained for future use.

Remember that calling PROTE$ only affects the LFT entry and that no physical change on the file is effected until CLOSE$ is called.

Entry point: 01105

Parameters:
B = LFN
C = new protection:
C0 = 1 for protection change
C6 = 1 for write protection
C7 = 1 for delete protection
DE = new LRN limit field; 0 for no change

Exit conditions: B = LFN; other registers indeterminate

Traps: none

51.6.8 POSIT$ - Position to a Record within a File

POSIT$ positions the file logically to the user LRN given. If the LRN given is -1, the current value in the LFT is used for positioning and the LFT entry is not changed. Positioning to LRN zero performs a logical "rewind" of sequential files.

Entry point: 01110

Parameters:
```
B  = LFN
DE = LRN (use LRN from LFT if DE = -1)
```

Exit conditions:
```
B = LFN
D = Physical Disk Address MSB
E = Physical Disk Address LSB
ZERO FALSE:  DE are valid, position was valid
ZERO TRUE:   DE are invalid, specified sector not
             in allocated space
other registers indeterminate
```

Traps: none

51.6.9 READ$ - Read a Record into the Buffer

READ$ causes the record pointed to by the LRN in the LFT entry specified by the LFN given, to be transferred from the disk to the disk controller buffer that corresponds to the LFN given. The LRN is incremented by one after the read if it was successful. If a parity fault is detected READ$ performs four to ten retries before giving a parity trap. Attempting to read a record that is not physically allocated will cause a RANGE trap.

Entry point: 01113

Parameters: B = LFN

Exit conditions:
```
B = LFN; other registers indeterminate
LRN = LRN + 1 if successful
```

Traps:

RANGE	LRN specified was out of range.
RPARIT	Record was unreadable.
FORMAT	PFN or LRN in record is incorrect.
OFFLIN	Drive is off-line.

51.6.10 WRITE$ - Write a Record from the Buffer

WRITE$ first takes the PFN and LRN values from the appropriate LFT and stores them into the first three bytes of the disk controller buffer that corresponds to the LFN given. It then transfers that buffer to the disk sector specified by the LRN. The LRN is incremented after the write if it is successful. If a parity fault is detected, WRITE$ tries up to ten times to obtain a good write before giving an error indication.

If WRITE$ tries to write a record beyond the space already allocated to the file, it will automatically attempt to allocate more space. If the space is available, it is allocated and the write occurs. If there is no more physical space on the disk or if there are no more entries in the RIB available for the new segment descriptor, a SPACE trap occurs.

Entry point: 01116

Parameters: B = LFN

Exit conditions: B = LFN; other registers indeterminate
LRN = LRN + 1 if successful

Traps:

WVIOLA	File is write protected.
WPARIT	Write/verify failure occurred.
OFFLIN	Drive is off-line.
RANGE	LRN specified was less than zero.
SPACE	Explained above.

51.6.11 GET$ - Get the Next Buffer Character

The LFT contains an entry called BUFADR (not to be confused with location 026 used by GETNCH) which points to a character in the disk controller buffer that corresponds to the given LFN. Each buffer contains 256 characters, but since the system uses the first three bytes in each sector to store the PFN and the LRN of each record, the user has only 253 bytes available.

Whenever READ$, WRITE$, or POSIT$ are executed, they set BUFADR to point to the third byte in the disk controller buffer (by setting the BUFADR field of the LFT entry to a three). Whenever GET$ is called, the byte pointed to by this pointer is fetched from the disk controller buffer and the pointer is incremented. If the byte being returned is not a valid user data byte (that is, BUFADR was 0, 1, or 2 on entry) then Carry is true on return, and register A contains the specified byte of the

buffer (which will be the PFN or one of the LRN bytes). The next buffer is not read automatically from the disk; the pointer simply ends-around. Upon the first call of GET$ which returns Carry true, the PFN will be obtained since it is contained in buffer location zero. A byte may also be accessed by simply setting BUFADR to the desired location.

Entry point: 01121

Parameters: B = LFN

Exit conditions: A = the byte obtained from the buffer
all other registers preserved
Carry true if location 0, 1, or 2 accessed

Traps: none

51.6.12 GETR$ - Get an Indexed Buffer Character

GETR$ is similar to GET$ except that it uses the logical buffer address supplied in the C register instead of the physical buffer address in the LFT for the address of the disk buffer byte to return. Calling GETR$ has no effect on the buffer pointer kept in the LFT. The physical buffer location is obtained by adding three to the value given in the C register to skip past the system data in the first three bytes in the disk buffer. Thus the user is presented with a logical space within a record that is addressed from 0 through 252. Normally, GETR$ exits with the value in the C register incremented by one and the carry condition false. However, if the C register is between 253 and 255 (inclusive) upon entry, it will not be incremented and exit will be made with the Carry condition true. In either case, the buffer byte located by the C register value plus three is returned in the A register. Therefore, the user may obtain any buffer byte with GETR$ but must remember to supply an address which is the physical buffer address minus three and remember not to assume that the C register will be incremented if he plans to access one of the first three physical bytes.

Entry point: 01124

Parameters: B = extended LFN
C = buffer location

Exit conditions: A = byte obtained
C = C + 1 if carry false
Carry true if 252 < C < 256

all other registers preserved

Traps: none

51.6.13 PUT$ - Store into the Next Buffer Position

PUT$ is similar to GET$ except that the byte presented in the A register on entry is stored into the buffer. Also, on return register A contains the physical address of the next byte to be accessed in the disk buffer. Carry is true if the byte stored was placed into the last physical location in the buffer. In standard, EDIT-format records, the last two bytes (at least) of the buffer are not used, and an 03 occurring earlier in the sector must indicate logical end-of-sector. (A complete description of the format for DOS text files can be found in the appendix describing Disk Data Formats.)

Entry point: 01127

Parameters: A = the byte to be stored in the buffer
B = LFN

Exit conditions: A as described above (physical address of next byte)
all other registers preserved
Carry true if location 255 was the destination byte.

Traps: none

51.6.14 PUTR$ - Store into an Indexed Buffer Position

PUTR$ is identical to GETR$ except that the byte presented in the A register is stored into the buffer.

Entry point: 01132

Parameters: A = byte to be written
B = extended LFN
C = logical buffer location

Exit conditions: C = C + 1 if carry false
Carry true if 252 < C < 256
all other registers preserved

Traps: none

51.6.15 BSP$ - Backspace One Physical Sector

BSP$ decrements the LRN in the LFT entry specified by the LFN given and then executes POSIT$. No check is made to prevent BSP$ from backing into a RIB. However, if one calls BSP$ and attempts to backspace back beyond system LRN 0 (user LRN -2, which is the master RIB) ZERO TRUE will be returned (as for POSIT$).

Entry point: 01135

Parameters: B = LFN

Exit conditions: B = LFN; other registers indeterminate
ZERO FALSE: valid backspace
ZERO TRUE: invalid backspace (attempt to backspace past master RIB)

Traps: none

51.6.16 BLKTFR - Transfer a Block of Memory

BLKTFR moves the number of bytes specified in the C register (0 causes transfer of 256 bytes) from the memory locations starting from the address in HL to the memory locations starting at the address in DE. Since exit is made with HL and DE pointing after the last byte moved and C equal to zero, transfers of more than 256 bytes may easily be made. Set C to the residual number of bytes to be moved (number of bytes modulo 256), call BLKTFR to move the residual number of bytes, then call BLKTFR again enough times to move the necessary number of 256-byte blocks. For example:

```
HL      SOURCE    Point to source string
DE      DEST      Point to destination string
LC      25        Set residual number
CALL    BLKTFR    Move 25 bytes
CALL    BLKTFR    Move 256 bytes
CALL    BLKTFR    Move 256 bytes
```

will cause 537 bytes to be transferred from SOURCE to DEST.

Entry point: 01143

Parameters: C = number of bytes to be moved
(0 moves 256 bytes)
HL = source address
DE = destination address

Exit conditions: HL = HL + C (HL + 256 if C = 0)
DE = DE + C (DE + 256 if C = 0)
B = unchanged
C = zero

Traps: none

51.6.17 TRAP$ - Set an Error Condition Trap

There are eight non-fatal error conditions, concerning the disk operating system file handling facilities, that may be trapped by the user. If the trap corresponding to a certain error is not set by TRAP$, the system displays a pertinent message and reloads the system if the trap occurs. If the trap is set, control is transferred to the address specified when the trap was set, with the subroutine return address stack in the state it had before the calling of the file handling routine that caused the error condition.

The only disk errors that cannot be trapped are those associated with the system tables on the disk. The occurrence of these errors causes the message

FAILURE IN SYSTEM DATA

to be displayed. Other errors that cannot be trapped have to do with the LFT entry not being open when a routine which tried to use data from the entry was called, invalid logical file numbers, invalid drive numbers, invalid trap numbers, and invalid physical file numbers.

If a trap occurs during a call to READ$ or WRITE$, the LRN in the LFT is not incremented; if the user wishes to continue processing records past the one which caused the trap, he must increment the LRN in the LFT himself.

TRAP$ sets the trap whose number is given in the C register to the address supplied in the D register (MSB) and E register (LSB). The trap is cleared by calling TRAP$ with D and E equal to zero. The trap is also cleared when the error condition occurs, at which time the B register will be loaded with the Logical File Number involved and control transferred to the indicated address.

In the following table, the mnemonic given after the trap number is the one used in the previous routine explanations. The capitalized lines are the messages displayed if the trap is not set.

0 - RPARIT - PARITY FAILURE DURING READ
A parity fault while reading a data record causes this trap.

1 - WPARIT - PARITY FAILURE DURING WRITE
A parity fault while writing a data record causes this trap.

2 - FORMAT - RECORD FORMAT ERROR
The physical file number or logical record number in the record read not matching the ones contained in the logical file table entry causes this trap. The physical position of a record is obtained from information in the Retrieval Information Block (RIB) and the PFN and LRN in the record are only checked to ensure that the drive is functioning correctly and that the user is not trying to read a record he has not written. This trap has nothing to do with the format of the 253 data bytes provided to the user.

3 - RANGE - RECORD NUMBER OUT OF RANGE
During a read, an access below system LRN zero or to a record above the currently allocated space causes this trap. During a write, an access below system LRN zero causes this trap.

4 - WVIOLA - WRITE PROTECT VIOLATION
An attempt to write on, delete, or shorten a file with the write protection bit set causes this trap.

5 - DVIOLA - DELETE PROTECT VIOLATION
An attempt to delete or shorten a file with the delete protection bit set causes this trap.

6 - SPACE - FILE SPACE FULL
An attempt to allocate more space when either the disk is full or no more segment descriptor slots in the RIB are available causes this trap.

7 - OFFLIN - DRIVE OFF LINE
An attempt to use a drive that is either physically absent or not online causes this trap.

The causes given for the various traps are the causes for DOS to issue the appropriate messages. Some of the DOS Command programs also cause the issuance of some of these messages for related reasons. For example, several DOS Utilities indicate a RECORD FORMAT ERROR if the sector formatting of a file being processed does not follow GEDIT (or DOS EDIT) standards. In such

cases the above details are sometimes not valid descriptions of the problem; in this example the 253 data bytes encountered may be the cause of the record format error.

In addition, FORMAT and RANGE traps are frequently the result of sequentially reading or otherwise processing a file which has no valid EOF, resulting in the program reading past the logical end of the file.

Entry point: 01146

Parameters: DE = trap address
C = trap number

Exit conditions: register contents indeterminate

Traps: none

51.6.18 EXIT$ - Reload the Operating System

EXIT$ closes any logical files (one through three) that are open and then reloads the operating system. EXIT$ is the normal exit for all DOS programs.

If MCR$ (01400) contains exactly two forward arrows ">>" followed by a command line, followed by an 015, the command interpreter will be reloaded, and the command line in MCR$ will be scanned and executed.

Consider a program that is to terminate and begin a predefined procedure. If the procedure is a chain procedure file PROC1/TXT, the program can simply place the following string in memory starting at MCR$:

```
>>CHAIN PROC1(015)
```

then, JMP EXIT$. This method is easier to work with than a series of programs linked by LOAD$ or RUN$, and has the advantage of reloading DOS, so its routines are available for later programs. If a program is set for automatic execution (via the AUTO command) the auto-execute will be performed before the command line left in MCR$ will be acted upon.

Entry point: 01151

Parameters: none

Exit conditions: no exit

Traps: none

51.6.19 ERROR$ -- Reload the Operating System

ERROR$ is identical to EXIT$ in all respects except that jumping to ERROR$ will abort an active CHAIN (refer to the CHAIN command in this manual for more details). A user program would exit through ERROR$ if an error of severity suggesting aborting a CHAIN occurred.

Entry point: 01140

Parameters: none

Exit conditions: no exit

Traps: none

51.6.20 WAIT$ -- DOS Wait-a-While "NOP" Routine

This routine, after being called, returns with all registers, condition codes, and the stack preserved; in effect a "NOP". Normally, the return is immediate. This routine should be used in loops which wait for conditions that are not time-critical to occur (for example, waiting for the keyboard operator to release the DISPLAY key). I/O status, including in particular the device addressed, is subject to change on return.

Entry point: 01170

Parameters: none

Exit Conditions: registers and condition codes unchanged

Traps: None

51.7 Keyboard and Display Routines

51.7.1 DEBUG$ - Enter the Debugging Tool

The debugging tool enables the programmer to load files by number, examine and modify memory locations, set breakpoints, and execute sections of his program. This facility greatly simplifies the task of debugging machine language programs.

The debugging tool can be entered from the command interpreter by entering a single pound sign (#) on the command line or from the user' s program by jumping to the entry point. When debug is executing, two numbers are displayed vertically in the last column of the screen. The five-digit top number is an address and the three-digit bottom number is the contents of that address. After these numbers are displayed, input is requested from the keyboard as indicated by a flashing cursor. Commands to the debugger are given in the form <n>X where <n> is any number of octal digits and X is a command character. The command is executed immediately upon depression of the command character key without waiting for the ENTER key (the ENTER character is a command in itself).

All keys that are not recognized are ignored, with a beep signaling the rejection. The BACKSPACE key is ignored, but since commands use only the lower eight or sixteen bits of <n>, errors in the entry of numbers can be corrected by striking several zeros and then entering the correct digits. Alternatively, the CANCEL key causes the current input line to be erased without changing the current address. Although display stops if the cursor runs off the screen during input, characters are still accepted.

The debugger maintains a current address that is usually displayed as the five digit number at the right of the screen. There are times, however, when the five digits at the right of the screen do not reflect the current address and caution must be exercised to avoid confusion as to the value of the current address. The ENTER key is normally used to change the current address, but depressing it without preceding it with any digits will cause the current address to be displayed. Therefore, if there is any doubt about the number being displayed on the screen, simply depressing the ENTER key will ensure that the current address is being displayed.

Whenever the debugger is entered either from a jump to the entry point or from a return from a breakpoint or call command, a

beep is given and the state of all of the alpha mode registers and condition flags is saved. The value initially displayed is the top of the stack at entry, unless DEBUG was entered from a DOS DEBUG breakpoint; in this case the address displayed is the address where the breakpoint was set. In all cases, the stack is preserved as at entry and the current address is set to the address displayed at entry. This display enables the user to observe the state of his program when the debugger was entered. Whenever a memory location is called or jumped to, the state of all of the alpha mode registers and condition flags is restored from the values saved at entry. Since these values are saved in memory, the programmer can simply modify these locations to change the values used to initialize the state of the alpha machine before control is transferred.

A major debugging technique is to set breakpoints at critical places in the program and execute portions of the program while checking the values of the registers and critical memory locations at each break. The debugger sets a breakpoint by storing a jump instruction to a special entry point within itself in the current address and the following two locations. (Notice that setting breakpoints less than three bytes apart is therefore not a good idea.) Before the jump is stored, the contents of the memory locations to be used are saved in a table in the debugger. When the breakpoint is reached, the memory locations are restored with their original contents. A maximum of four breakpoints may be active at any one time. A command is provided to insure that all breakpoints have been restored. When a breakpoint is executed, the current address is set to the first byte of the breakpoint jump instruction. Since the J command causes a jump to the current address if no digits precede it, one can continue execution of the routine that was broken by simply depressing the J key. Execution will continue with the first byte that was overstored by the breakpoint jump with the state of the alpha machine exactly like it was before the break occurred. Thus, the programmer can set a breakpoint, start execution, examine the registers when the break occurs (since register viewing does not change the current address) and then depress the J key to continue execution. This technique allows him to practically single step his program.

ENTRY POINT: 01154

COMMANDS:

B - Set a breakpoint at the location given or, if no number is given, at the current address. Caution should be exercised to insure that the current address is pointing to the desired

location if it is used.

C - Execute a call to the number given or, if no number is given, to the current address. The alpha machine state is loaded from the values saved in the debugger before the call is executed. A return to the call causes the debugger to be re-entered and the alpha machine state to be saved.

D - Decrement the current address (any digits given are ignored).

G - Get the physical file specified from the disk. Care must be exercised that a file is not loaded that will overlay the debugger (locations 0-01377 and 06000-07377). If the file does not exist or contains a record of illegal loader format, a beep will be given. The first digit of the last four entered is the logical drive number from which the file is to be loaded. The following three digits are the physical file number. For example, 02003G will load SYSTEM3/SYS from drive two. To load PFN 0115 from drive 0, simply enter 115G.

I - Increment the current address (any digits given are ignored).

J - Execute a jump to the number given or, if no number is given, to the current address. The alpha machine state is loaded from the values saved in the debugger before the jump is executed.

M - Modify the contents of the current address. The least significant eight bits of the octal number given before the command character are used for the new memory value. If no digits are given, a zero is assumed.

P - Turn on the P-counter display (to the left of the current address). This display is a foreground driven routine which takes the value of the P-counter when the interrupt occurred and displays it vertically. This implies that the value shown is the background P-counter at 32 millisecond sample points. When the display is active, simultaneous depression of the KEYBOARD and DISPLAY keys will cause the debugger to be entered regardless of what is currently being executed in the background. When such entry occurs, the current address points to the location where the background program was interrupted so that execution can be resumed with the J command.

R - Display the saved alpha mode register value. The registers are referenced by number (0-A, 1-B, 2-C, 3-D, 4-E, 5-H, 6-L, and 7-Conditions). The condition code is stored with bits 7=Carry, 6=Sign, bits 5 through 2 always zero, 1=(-Zero and

-Sign), and 0=(-Zero and -Parity). (The easiest way to understand this is to realize that the condition code as displayed, added to itself, results in restoring all four conditions to their entry values.) When a register is displayed, the address shown is the memory location used to store the value of that register. This does not, however, affect the current address. The registers may be initialized for a C or J command by simply storing into the memory locations displayed when the registers are displayed.

X - Turn off the P-counter display.

- Clear all breakpoints. The current address will reflect the location of the last point cleared.

. - Perform the M command followed by the I command.

CANCEL - Erase the entered number without changing the current address.

ENTER - Change the current address to the digits entered. If no digits are entered, the current address in effect will be displayed.

51.7.2 KEYIN$ - Obtain a Line from the Keyboard

KEYIN$ obtains a string of characters from the keyboard, displaying them on the screen and storing them in memory as they are entered. When KEYIN$ is called, the cursor is turned on and characters requested. Backspacing off the beginning of the line, entering more than the specified maximum number of characters, or running off the screen is prevented. The routine turns off the cursor and returns when the ENTER key is depressed.

Entry point: 01157

Parameters: C = maximum number of characters accepted (including ENTER)
D = initial horizontal cursor position
E = vertical cursor position
HL=> input buffer

Exit conditions: String terminated by 015
HL=> 015 in input buffer
D = horizontal position of ENTER
E = unchanged
C = 0
B = undefined

51.7.3 DSPLY$ - Display a Line on the Screen

DSPLY$ displays a string of characters stored in memory onto the screen. Certain characters denote control functions according to the following table:

003 - end of string
011 - new horizontal position follows
013 - new vertical position follows
015 - end of string with CR/LF
021 - erase to end of frame
022 - erase to end of line
023 - roll up one line

If the string to be displayed starts with either or both horizontal or vertical cursor controls, then either or both of the corresponding values need not be in D or E at entry. If the cursor is not positioned on the screen with DE or 011 and 013 the results of 021, 022, or 023 are undefined.

Entry point: 01162

Parameters: D = initial horizontal cursor position
E = initial vertical cursor position
HL => string in memory

Exit conditions: DE = cursor position after the last character displayed
HL => byte after the string terminator
A,B,C undefined

CHAPTER 52. DOS FUNCTION FACILITY (DOSFNC)

The page of memory located between 07400 and 07777 contains a special loader and overlay area. This "loader" can load any one of up to 255 DOS overlays, each up to 124 bytes long. The loader resides in the first half of the page and the overlays all load into the second half of the same page. The overlays reside on disk in physical file 7, called SYSTEM7/SYS. The design of the DOS FUNCTION loader is such that overlays are loaded only if necessary; that is, if the same overlay is called several times in sequence, it is not reloaded each time. The overlays provide the DOS assembly language programmer with many useful utility functions. Parameterization of DOS FUNCTIONs varies with the individual functions, the only basic requirement being that upon entry to the DOS FUNCTION loader, the A register contains the function number (1-255). Use of functions not yet installed will produce indeterminate results, such as format traps, range traps, processor halts, and the like. DOS FUNCTIONs are normally loaded from the SYSTEM7/SYS on the booted drive, or may come from the memory resident overlays if available (see UTILITY/OVL).

Upon the first call to DOSFNC (the DOS FUNCTION loader), SYSTEM7/SYS is opened as LF0 and the LFT entry saved within the DOS FUNCTION loader. Upon subsequent calls to DOSFNC, the entry is simply moved back into the LFT, eliminating the need to re-open SYSTEM7/SYS each time a function is loaded. The file is only closed by reloading DOS, either by depressing RESTART or by a program passing control to BOOT$, EXIT$, or ERROR$.

Since new DOS functions will be added as necessary the following descriptions should not be considered exhaustive.

Entry point: 07400

Parameters: A = Function number (1-0377)
Others required by individual functions

Exit conditions: Defined separately for each function.

52.1 FUNC1 - Retrieve Directory and C.A.T. Addresses

Uniform attributes for all subfunctions

```
On entry,     A = function number (1)
              C = subfunction number (0,1,2,3,4,5,6,7)
On exit,      B,C,H,L all unchanged
              CARRY FALSE:  successful completion
              CARRY TRUE:   invalid subfunction number
```

All other entry/exit parameters and conditions are described seperately for each individual subfunction.

DOS FUNCTION: 1 SUBFUNCTION: 0

Return the address of a specified directory sector in DE.

```
On entry,     B = directory sector number (0-15) OR
              PFN of entry in the directory sector
On exit,      A indeterminate
              DE = PDA of specified directory sector.
```

DOS FUNCTION: 1 SUBFUNCTION: 1

Return the two-byte physical disk address for each of the 16 master directory sectors, into a 32-byte work area provided by the user.

```
On entry,     HL => 32-byte work area to receive the PDA's
On exit,      all registers preserved
              User-provided work area contains 16 PDA's,
              one corresponding to each prime directory
              sector, in ascending order (LSB,MSB).
```

DOS FUNCTION: 1 SUBFUNCTION: 2

Return the two-byte physical disk address of each of the 16 directory sector backups, in ascending order, into a 32-byte user-provided work area.

```
On entry,     HL => 32-byte work area to receive the PDA's
On exit,      all registers preserved
              User-provided work area contains 16 PDA's,
              one corresponding to each backup directory
              sector, in ascending order (LSB,MSB).
```

DOS FUNCTION: 1 SUBFUNCTION: 3

Return the physical disk address of the Cluster Allocation Table (CAT) in the DE register pair.

```
On entry,     no further conditions
On exit,      A indeterminate
              DE = PDA of prime CAT
```

DOS FUNCTION: 1 SUBFUNCTION: 4

Return the physical disk address of the backup Cluster Allocation Table (CAT) in the DE register pair.

```
On entry,     no further conditions
On exit,      A indeterminate
              DE = PDA of backup CAT
```

DOS FUNCTION: 1 SUBFUNCTION: 5

Return the physical disk address of the lockout CAT.

On entry, no further conditions
On exit, A indeterminate
DE = PDA of lockout CAT

DOS FUNCTION: 1 SUBFUNCTION: 6

Return the physical disk address of the lockout CAT backup.

On entry, no further conditions
On exit, A indeterminate
DE = PDA of lockout CAT backup

DOS FUNCTION: 1 SUBFUNCTION: 7

Return the address of a backup directory sector (in DE).

On entry, B = backup directory sector number (0-15)
or PFN of a file entry contained therein
On exit, A indeterminate
DE = PDA of backup directory sector

52.2 FUNC2 - Retrieve Directory Sector or Filename

Uniform attributes for all subfunctions

```
On entry,     A = function number (2)
              C = subfunction number (0,1,2)
On exit,      all registers preserved
              CARRY TRUE: error or invalid subfunction number
```

All other entry/exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 2 SUBFUNCTION: 0

Read in the directory sector containing the 16-byte directory entry corresponding to the PFN given, on a specified logical drive.

```
On entry,     B = LFN as per DOS standard; (0, 16, 32, 48)
              D = PDN (logical drive number of file)
              E = PFN
On exit,      CARRY FALSE: Selected directory sector is in
                           buffer specified, which is the
                           selected buffer upon exit.
              CARRY TRUE:  I/O error, further defined
                           as follows:
              ZERO FALSE:  Specified drive is off-line
              ZERO TRUE:   Unable to read sector due to CRCC
                           error during read, or unrecoverable
                           failure to find sector.
```

DOS FUNCTION: 2 SUBFUNCTION: 1

Get the 16-byte directory entry corresponding to a specified PFN on a given logical drive.

```
On entry,     B = LFN as per DOS standard; (0, 16, 32, 48)
              D = PDN (logical drive number of file)
              E = PFN
              HL => 16-byte area to receive the entry
```

On exit,
- CARRY FALSE: Entry is in user's area.
- CARRY TRUE: I/O error, further defined as follows:
  - ZERO FALSE: Specified drive is off-line.
  - ZERO TRUE: Unable to read sector due to CRCC error during read, or unrecoverable failure to find sector.

DOS FUNCTION: 2 SUBFUNCTION: 2

Get name/ext (pfn) for a specified numbered file on a specified logical drive. (Same basic format as used by DOS CAT command).

On entry,
- B = LFN as per DOS standard; (0, 16, 32, 48)
- D = PDN (logical drive number of file)
- E = PFN
- HL => 20 byte area to receive the entry

On exit,
- CARRY FALSE: User's 20-byte area contains the name, extension and PFN of the specified file, for example: EDIT/CMD (037) where the right parenthesis is followed by an 003.
  UNLESS ZERO TRUE:
  Implies that the file number specified does not exist.
- CARRY TRUE: I/O error, further defined as follows:
  - ZERO FALSE: Specified drive is off-line.
  - ZERO TRUE: Unable to read sector due to CRCC error during read, or unrecoverable failure to find sector.

NOTICE: The use of this subfunction only (of those in DOS FUNCTION 2) requires that the DOS command interpreter be present (the command interpreter resides from 013400-017000).

52.3 FUNC3 - Retrieve RIB Information

Uniform attributes for all subfunctions:

```
On entry,     A = function number (3)
              C = subfunction number (0,1,2,3)
```
All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 3 SUBFUNCTION: 0

Return the number of sectors allocated to a file on disk.

```
On entry,     B = drive number (same as C as provided for OPEN$)
              DE = proper OPEN$ parameters defining the file
                   to be accessed.
On exit,      CARRY FALSE: successful completion
                           HL = length of file (MSB,LSB) in sectors
                           RIB for file specified is in LF0
                           disk buffer.
              CARRY TRUE:  Error occurred, any one of:
                           OPEN failed on file specified;
                           unable to read RIB;
                           parity or drive off-line.
```

DOS FUNCTION: 3 SUBFUNCTION: 1

Get the RIB for a specified file into the LF0 disk buffer.

```
On entry,     B = drive number (same as C as provided for OPEN$)
              DE = proper OPEN$ parameters defining the file
                   to be accessed.
On exit,      all registers preserved
              CARRY FALSE: successful completion
                           RIB for file specified is in
                           LF0 disk buffer.
              CARRY TRUE:  Error occurred, any one of:
                           OPEN failed on file specified;
                           unable to read RIB;
                           parity or drive off-line.
```

DOS FUNCTION: 3 SUBFUNCTION: 2

Read a RIB for a file, given the first two bytes of the directory entry.

```
On entry,     B = drive number (same as C as provided for OPEN$)
              D = RIB pointer, (MSB) from directory, or LFT
              E = RIB pointer, (LSB) from directory, or LFT
On exit,      all registers preserved
              CARRY FALSE: successful completion
                           RIB for file specified is in
                           LFO disk buffer
              CARRY TRUE, ZERO FALSE: Specified drive is off-line.
              CARRY TRUE, ZERO TRUE:  Parity error occurred during read.
```

DOS FUNCTION: 3 SUBFUNCTION: 3

Return segment descriptor information from a RIB.

```
On entry,     RIB is in LFO disk buffer.
              BUFADR field in LFO LFT entry points to
              segment descriptor
On exit,      LFO buffer unchanged.
              CARRY TRUE:  successful completion
                           A = starting cyl. number for segment
                           B = starting cluster number for segment
                           DE = number of sectors in the segment
                           BUFADR points to next segment
                             descriptor.
                           RIB undisturbed
              CARRY FALSE: BUFADR pointed after logical
                           end of RIB;
                           BUFADR contents undefined.
```

52.4 FUNC4 - Retrieve DOS Configuration Information

Uniform attributes for all subfunctions:

```
On entry,     A = function number (4)
              C = subfunction number (0,n)
On exit,      A = DOS configuration value
              CARRY FALSE: successful completion
              CARRY TRUE:  possibly invalid subfunction number
```

Different subfunction numbers return different DOS configuration bytes. These values, returned in A, are numeric items which change in value depending upon which DOS is running. The subfunction numbers, along with the significance of the returned value, are:

```
0 - Letter of this DOS (A,B,C,D,etc.)
1 - DOS Version ('2' typ.)
2 - DOS Revision ('5' typ.)
3 - Total number cylinders on disk (203 typ.)
4 - Maximum Logical Drive (3,15 typ)
5 - Year of Compilation (79 typ)
6 - Day of Compilation (130 typ)
7 - Cluster Mask (0340 typ)
8 - Increment Cluster number (040 typ)
9 - Sector Mask (037 typ)
10 - Maximum Sector Number in PDA (23 typ)
11 - Number of Sectors/Cluster (3,6,24 typ)
12 - Number of Clusters/Cylinder (4,8 typ)
13 - Number of Clusters/Track (1,4 typ)
14 - Number of Functions in SYSTEM7 (24 typ)
15-17 (Unused)
18 - DOS Pre-release (040 if released)
19 - DOS Maintenance Release (040 if none)
20-24 (Unused)
25 - Get VOLID address into (DE)
26 - no longer valid
```

52.5 FUNC5 - Request Access to System Tables

This function is used when running under the Partition Supervisor (PS). This function must be called before and after any changes are made to the system tables on any drive.

Uniform attributes for all subfunctions:

```
On entry,     A = function number (5)
              C = subfunction number (0,1)
```

All other entry and exit parameters and conditions are described seperately for each individual subfunction.

DOS FUNCTION: 5 SUBFUNCTION: 0

Request exclusive update permission to system table sectors on disk.

```
On entry,     D = physical drive (PDN) of drive
On exit,      CARRY FALSE: successful completion
                           Exclusive use of specified drive
                           guaranteed.
              CARRY TRUE:  Error occurred.
```

DOS FUNCTION: 5 SUBFUNCTION: 1

Release exclusive update authority for system table sectors on disk.

```
On entry,     D = physical drive (PDN) of drive
On exit,      CARRY FALSE: successful completion
                           Exclusive use of specified drive
                           released.
              CARRY TRUE:  Error occurred.
```

52.6 FUNC6 - Keyboard / Display Interface Routines Function

Uniform attributes for all subfunctions:

```
On entry,     A = function number (6)
              C = subfunction number (0-11)
On exit,      CARRY TRUE:   illegal subfunction
```

All other entry and exit parameters and conditions are described sepatately for each individual subfunction.

DOS FUNCTION: 6 SUBFUNCTION: 0

Check the status of the KEYBOARD and DISPLAY keys.

```
On entry,     no further conditions
On exit,      SIGN TRUE:     KEYBOARD key pressed
              PARITY TRUE:   DISPLAY key pressed
              all registers preserved
```

DOS FUNCTION: 6 SUBFUNCTION: 1

Check for character ready.

```
On entry,     no further conditions
On exit,      ZERO TRUE:     No character present
              ZERO FALSE:    Ready to get character
              all registers preserved
```

DOS FUNCTION: 6 SUBFUNCTION: 2

Get a character from the Keyboard.

```
On entry,     DE = horizontal, vertical screen coordinates
On exit,      ZERO TRUE:   no character present
              all registers preserved
              ZERO FALSE:   character in the A register
              all other registers preserved
```

DOS FUNCTION: 6 SUBFUNCTION: 3

Write the character in the B register to the screen.

```
On entry,     DE = horizontal, vertical screen coordinates
```

```
                B = character to be written to screen
On exit,        CARRY TRUE:  D or E out of range
                all registers preserved
                CARRY FALSE:  character written
                all registers preserved
```

DOS FUNCTION: 6 SUBFUNCTION: 4

Return the HOME-UP position in DE.

```
On entry,       no further conditions
On exit,        DE = address of top line / left column of CRT
                all other registers preserved
```

DOS FUNCTION: 6 SUBFUNCTION: 5

Return the HOME-DOWN position in DE.

```
On entry,       no further conditions
On exit,        DE = address of bottom line / left column of CRT
                all other registers preserved
```

DOS FUNCTION: 6 SUBFUNCTION: 6

Turn on the Cursor.

```
On entry,       DE = horizontal, vertical screen coordinates
On exit,        all registers preserved
```

DOS FUNCTION: 6 SUBFUNCTION: 7

Rollup the screen 1 line.

```
On entry,       DE = horizontal, vertical screen coordinates
On exit,        all registers preserved
```

DOS FUNCTION: 6 SUBFUNCTION: 8

Erase from cursor position to end of frame.

```
On entry,       DE = horizontal, vertical screen coordinates
On exit,        all registers preserved
```

DOS FUNCTION: 6 SUBFUNCTION: 9

Erase from cursor position to end of line.

On entry, DE = horizontal, vertical screen coordinates
On exit, all registers preserved

DOS FUNCTION: 6 SUBFUNCTION: 10

Rolldown the screen 1 line.

On entry, DE = horizontal, vertical screen coordinates
On exit, CARRY TRUE: illegal operation for this device
all registers preserved
CARRY FALSE: screen rolled down 1 line
all registers preserved

DOS FUNCTION: 6 SUBFUNCTION: 11

Turn off the Cursor.

On entry, DE = horizontal, vertical screen coordinates
On exit, all registers preserved

52.7 FUNC7 - Test the Disk Buffer Memory

Disk buffer memory test function.

This DOS FUNCTION performs a rotating, cycling test of the disk controller buffer memories. It returns upon the keyboard becoming READ READY, or upon encountering a buffer failure, whichever occurs first.

On entry, no special conditions
On exit, all registers preserved
ZERO TRUE: buffer memory test normal
ZERO FALSE: failure in buffer memories

52.8 FUNC8 - Timed Pause

Pause function.

This DOS FUNCTION provides the user program with a timed pause. The requested pause may be up to approximately four hours long.

On entry, B = foreground process number to use (0-7)
CDE = number of milliseconds to pause
(C = most signifigant, E = least signifigant)
On exit, all registers preserved

Note that if foreground process numbers 4-7 are used, the wait time is effectively multiplied by four, allowing a maximum wait time in excess of eighteen hours. Also note that the time required to start up the DOS FUNCTION is not considered part of the time paused. Since the DOS FUNCTION may or may not be resident when called, this function may wait longer than the quantity in CDE and therefore must not be used for timing extremely critical, short term intervals.

52.9 FUNC9 - Non-Sharable Resource Status Request

This DOS FUNCTION is used to allocate and de-allocate a system resource. Typically, this function is used when a program is going to run under the Partition Supervisor (PS). The use of this function will prevent conflicting use of I/O devices by the programs running in the two partitions. For example, the DOS utilities that use the printer (LIST, FILES, etc.) all call this function before they use the printer. Then, if a DATASHARE print statement is executed in the other partition, the listings will not be intermixed.

Uniform attributes for all subfunctions:

On entry, A = function number (9)
C = subfunction number (0,1)

All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 9 SUBFUNCTION: 0

Request use of a non-shareable system resource (printer, tape drive, and so on).

On entry, B = Resource Number
0 - Local Line Printer
1 - Servo Printer
2 - (Un-defined)
3 - (Un-defined)
4 - Cassette Tape Decks
5 - 7 or 9 Track Tape
6 - Multiport Comm Box 1 (all ports)
7 - Multiport Comm Box 2 (all ports)

On Exit, CARRY TRUE, ZERO TRUE: Permission to use granted.
CARRY TRUE, ZERO FALSE: Error occurred.
CARRY FALSE, ZERO TRUE: Already allocated to same partition. In this case, the device may be used, but must not be deallocated when finished.
CARRY FALSE, ZERO FALSE: Already allocated to other partition.

DOS FUNCTION: 9 SUBFUNCTION: 1

Release non-sharable resource for use by next party. This subfunction should be called after a process receiving access to a resource using subfunction zero has received CARRY TRUE, ZERO TRUE return, and finishes using the resource it wanted to use.

The only status returned by subfunction one that is likely to change upon waiting is CARRY FALSE, ZERO FALSE. In this case, the program wishing to release the resource should wait, perhaps five seconds (use function 8), and then retry the request. Any other status is not subject to change.

This subfunction cannot be used to test for printer busy, since if an invoking program in the same partition had allocated the device, the test would release it, possibly resulting in losing the device to a competing partition.

Indefinite postponement can be prevented by always allocating non-sharable resources in descending numerical sequence (when more than one non-sharable resource is needed at the same time).

On entry, all parameters identical to those for subfunction 0
On exit, CARRY TRUE: error
CARRY FALSE, ZERO TRUE: released
CARRY FALSE, ZERO FALSE: Resource was in use by different partition, therefore not released.

52.10 FUNC10 - Partition Information Function

This function is used to qualify a program to run in a "fixed" partition under the Partition Supervisor (PS), and to provide DOS/PS partition configuration information.

Uniform attributes for all subfunctions:

```
On entry,        A = function number (10)
                 C = subfunction number (0,1)
```

All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 10 SUBFUNCTION: 0

Authorize invoking program to execute in a fixed type partition.

```
On entry,     C = subfunction number (0)
On exit,      no significant conditions
```

DOS FUNCTION: 10 SUBFUNCTION: 1

Provide DOS/PS configuration information.

```
On entry,     C = subfunction number (1)
On exit,      HL => configuration list
                    The list may not be modified and is
                    guaranteed only until the next call to any
                    system routine.  List format described below.
              BYTE 0:  partition ID
                       Space if not running under PS, otherwise,
                       a unique identifier.
              BYTE 1:  region size - in number of K (16, 48, etc)
              BYTE 2:  number of disk buffers (4, 16, etc)
              BYTE 3:  .... ...1 implies fixed partition
              BYTE 4:  multiport I/O bus address for console on port
                       (0 implies console on port not active)
              BYTE 5:  multiport port select code of console port
                       (only if byte 4 is non-zero)
```

52.11 FUNC11 RAM Screen Loader

Uniform attributes for all subfunctions:

On entry, A = function number (11)
C = subfunction number (0,1,2)

All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 11 SUBFUNCTION: 0

Load one or more character combinations into the RAM display character generator.

On entry, B = default first character to be loaded
HL = starting address of character set definition list

The list consists of consecutive entries of either five or six bytes each. The first byte, if present, indicates the 7-bit character combination whose bit pattern definition follows. The presence of the first byte is indicated by its sign bit being set. If the first byte of the first entry is not present, the 7-bit character value in the B register is used instead. The definition list may contain any mixture of six-byte and five-byte entries. The end of the list is indicated by an 0200. This implies that the bit combination displayed for a binary zero cannot be imbedded in a list, but can only appear at its beginning; null lists are not allowed. The five data bytes following represent the five columns of bits for each displayed character and can each have values of 0 (a blank column) to 0177 (a vertical line). The 0100 bit is at the top of the character displayed; the 1 bit is on the bottom row of the displayed character.

On exit, CARRY FALSE, ZERO FALSE: RAM display not present
CARRY FALSE, ZERO TRUE: normal completion
CARRY TRUE: error (should not occur)

DOS FUNCTION: 11 SUBFUNCTION: 1

Load a single character combination to RAM display.

On entry, B = default character to be loaded
HL = address of five-byte or six-byte bit pattern definition

The first byte, if present, takes precedence over the character indicated by the B register. Presence of the first byte is indicated by the sign bit being set.

On exit, CARRY FALSE, ZERO FALSE: RAM display not present
CARRY FALSE, ZERO TRUE: normal completion

DOS FUNCTION: 11 SUBFUNCTION: 2

Subfunction two requests reloading of the standard character set on program termination. Calling this subfunction will result in the standard DOS character set being reloaded upon the next entry to DOS$. Entry to the DOS at DOS$ is the result of transfer of control to EXIT$, BOOT$, ERROR$ as well as DOS$. Return to the DOS via NXTCMD, CMDAGN, and CMDINT do not result in the display being immediately reloaded, (but it eventually will be reloaded upon subsequent entry at DOS$ as described).

52.12 FUNC12 - Enable Memory Resident Overlays

This function enables memory resident overlays and DOS Functions if running on DOS.D, DOS.E, or DOS.G. The UTILITY/OVL file will be loaded and use of its memory resident overlay code will begin. If the memory resident overlays cannot currently be loaded (PSACT or NETACT bits of DOSFLAG set, or UTILITY/OVL not present) then this function performs no action.

On entry, A = function number (12)
C = subfunction number (0)

On exit, all registers indeterminate
$MEMU bit of DOSFL2 on if successful

52.13 Overlay Loader (FUNC-13,14,15)

DOS functions 13, 14 and 15 are used to load "overlay libraries". Using these functions, one need only have a single directory entry for a program and its associated overlays (called "members"). The overlay library format is described in detail in the library utility program User's Guide (LIBSYS), since that program is responsible for creating and maintaining libraries. Program libraries can be either absolute or relocatable code.

Function 13 performs a library lookup by name.

Function 14 actually performs the library load from an absolute library.

Function 15 performs the library load from a relocatable library.

Below is an example of the use of these DOS functions in loading an absolute program overlay. In the example, the "root" program has several overlays; the root program was invoked from the keyboard by entering "FLX/ABS".

```
.
.
.
          HL        LFT+LFO             Save opened LFT entry
          DE        SAVELFT             (FLX/ABS)
          LC        16                  16 bytes
          CALL      BLKTFR
.
.
. To lookup the member named "FLXOAW":
.
NAMLOD    LA        13                  Prepare to lookup by name
          DE        OVLNAM
          HL        SAVELFT
          CALL      DOSFNC              Lookup by name
          JTC       ABORT               True Carry is error.
          JUMP      LOAD                (DE contains LRN if Carry False)
.
. To actually load the absolute file (and execute it):
.
LOAD      LA        14                  Prepare to load the file
          HL        SAVELFT
          CALL      DOSFNC              Load starting at LRN (DE)
          JTC       ABORT               True Carry is error.
.                                       (HL is transfer address if Carry False)
          PUSH                          Push entry point onto stack
          RET                           and transfer control there
.
.
.
SAVELFT   SK        16                  LFT save area
OVLNAM    DC        'FLXOAW  '          8-byte name
.
.
.
Note: All lookups (FUNC13) should be done first, and then all
      loads (FUNC14 or FUNC15) so functions will not be reloaded
      as often.
```

52.14 FUNC-13 Overlay Lookup By Name

Return the LRN of library member <name>, pointed to by DE into DE.

```
On entry,     A = function number (13)
              DE => address of 8-byte file name
              HL => 16-byte save area of user opened LFT
                    (not LF0)
On exit,      CARRY TRUE:  name not found
              CARRY FALSE:
                A = library type (see LIBSYS User's Guide)
                BC = undefined
                DE = LRN (LSB,MSB) of library member
                HL = entry value of DE + 8
```

52.15 FUNC-14 Load Absolute Library Member

Load the absolute member beginning at LRN given in DE.

On entry, A = function number (14)
DE = LRN (LSB,MSB) of member to be loaded.
HL => 16-byte save area of user opened LFT (not LF0)

On exit, CARRY TRUE: unloadable file
CARRY FALSE:
A,B,C,D,E undefined
HL = transfer address of member

52.16 FUNC-15 Relocatable Loader

DOS Function 15 uses the file UTILITY/LNK to perform the actual loading procedure. This function uses memory from 06000 through 07377 in addition to the DOS Function loader area from 07400 through 07777. Uniform attributes for all subfunctions:

```
On entry      A = function number (15)
              B = LFN of opened relocatable library
              C = subfunction number (0 or 1)
```

All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 15 SUBFUNCTION: 0

Return the size of relocatable member into DE.

```
On entry      DE => LRN 0 of library member (from DOSFNC 13 typically)
On exit       CARRY TRUE:  invalid library format
              CARRY FALSE:
                C,H,L  undefined
                A = revision level of member
                    (for UTILITY/REL members only)
                B = entry value
                DE = program length (LSB,MSB)
```

DOS FUNCTION: 15 SUBFUNCTION: 1

Load a relocatable library member.

```
On entry      HL => address of parameter list (defined below)
              DE => LRN 0 of library member (from DOSFNC 13 typically)
On exit       CARRY TRUE:  link error
              CARRY FALSE:
                A,C = undefined
                B = unchanged
                DE = next available address
                HL = transfer address
```

Parameter table for DOS FUNCTION 15 SUBFUNCTION 1:

0-1 origin address (LSB,MSB)

2-3 address of external reference work area terminated by 000
0377,0377 implies no work area, that is no external references.
Example:

```
RPT     20
DC      'bbbbbbbb',*-1
DC      0
```

4-5 address of external definition work area terminated by 000
0377,0377 implies no work area, that is no external definitions.
Example:

```
RPT     20
DC      'bbbbbbbb',*-1
DC      0
```

6-7 LRN (LSB,MSB) of relocatable module

52.17 FUNC-16 Disable Memory Resident Overlays

This function disables usage of memory resident overlays (UTILITY/OVL) enabled by DOS Function 12. Uniform attributes for all subfunctions:

```
On entry,        A = function number (16)
                 C = subfunction number (0 or 1)
```

All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 16 SUBFUNCTION: 0

Stop using overlays.

```
On entry,        no special conditions
On exit,         all registers and conditions indeterminate
                 $MEMU bit of DOSFL2 cleared
```

DOS FUNCTION: 16 SUBFUNCTION: 1

Memory resident overlays overstored; stop using overlays. Caution: do not overstore system RAM (0160000 to 0167377) until after calling this function.

```
On entry,        no special conditions
On exit,         all registers and conditions indeterminate
                 $MEMU bit of DOSFL2 cleared
                 $MEMD bit of DOSFL2 set

                 all registers and conditions indeterminate
```

CHAPTER 53. CASSETTE HANDLING ROUTINES

Standard record formats, identifiers, and file marker record conventions on cassettes are established by the Cassette Tape Operating System (CTOS). Routines capable of dealing with cassettes in a manner compatible with CTOS are provided as part of the Disk Operating System to enhance its overall capability. For detailed information on cassette format and organization, see the Cassette Tape Operating System Manual.

All of the DOS cassette routines are foreground driven and are among the few routines within the DOS which make use of the foreground handling facility. Being foreground driven, however, does not alter the manner in which the routines are handled since all interfacing between the background and foreground is handled by the system. It does allow increased speed of operation with the cassettes since the user may be processing one record while another is being read from or written to the tape. This is evident in the way the DOS slews the tape when transferring information between it and the disk.

Some of the cassette handling routines initiate foreground action and then return immediately to the user while others wait for I/O completion. All of the routines wait for any uncompleted I/O to finish before starting something new. In the cases of reading or writing on the same deck, requesting the next operation before the completion of the first will cause the tape to automatically slew instead of stopping between records. Slewing occurs only in the case of a read followed by another read or a write followed by another write on the same deck. The only cases where caution must be exercised are in the read and write routines which return immediately after starting the I/O operation. If the user does not wait for transfer to complete, he might accidentally attempt to use the data before it is completely read or change the data before it is completely written. In the second case, records with incorrect parity will usually be generated. Routines are provided, however, which automatically wait for transfer to complete, relieving the user of concern for the fact that the routines are foreground driven if he has no need for the advantages.

The various error conditions associated with cassette handling can be trapped by the user. If a trap is not set and an error occurs, an error message similar to messages generated by CTOS will be displayed and the DOS reloaded. If the trap has been

set, execution will jump to the address specified and the trap will be cleared. The traps are identified in the error message by a letter similar to the CTOS identification. In the relevant cases, the same letter is used in the DOS as is used in the CTOS. In the following routine descriptions the relevant letter will be given in the 'Traps' section.

Most of the cassette routines are parameterized by a deck number given in the B register. This number is a zero for the rear deck and a one for the front deck. The cassette handler routines use interrupt slot 1 for their foreground process.

53.1 TPBOF$ - Position to the Beginning of a File

TPBOF$ positions the cassette in the specified deck to the specified file. The search for the file marker of the desired file is started with backward motion of the tape. If a marker of lower value than the file number requested or the beginning of the tape is encountered, the search will be reversed to forward motion of the tape. If then a marker of larger value than the file number requested, the end of the tape, or a record of unrecognizable format is encountered, an error G will be given. Otherwise, the file is left positioned before the first data record.

Entry point: 010000

Parameters: B = deck number
C = physical file number (0-0177)

Exit conditions: none

Traps: D Unrecognizable record found.
G File could not be found.

53.2 TPEOF$ - Position to the End of a File

TPEOF$ moves the tape forward until the next file mark is found. It then backspaces the tape one record to leave it at the end of the current file.

Entry point: 010005

Parameters: B = deck number

Exit conditions: none

Traps: D Unrecognizable record found.
E End of tape encountered.

53.3 TRW$ - Physically Rewind a Cassette

TRW$ rewinds the cassette on the selected deck by first slewing backwards to ensure that the tape is not on the trailer and then performing a hardware rewind.

Entry point: 010012

Parameters: B = deck number

Exit conditions: none

Traps: none

53.4 TBSP$ - Physically Backspace One

TBSP$ simply executes a hardware backspace function. No checking is performed on the data passed over. However, backspacing onto clear leader causes an end of tape trap.

Entry point: 010017

Parameters: B = deck number

Exit conditions: none

Traps: E Beginning of tape encountered.

53.5 TWBLK$ - Write an Unformatted Block

TWBLK$ writes the specified number of bytes (0 causes 256 to be written) from the memory buffer specified onto the cassette in the deck specified. Only the bytes specified will be written on the tape.

Entry point: 010024

Parameters: B = deck number
C = number of bytes to write (0 for 256)

HL => start of buffer

Exit conditions: none

Traps:
- E End of tape encountered.
- Z Premature deck ready status occurred.

53.6 TR$ - Read a Numeric CTOS Record

TR$ reads a record of CTOS numeric format into the memory locations specified. The length of the record is stored in the specified memory location and the data bytes are stored in the locations that follow. Return is made from TR$ as soon as the read operation is started but the user cannot use the data until the operation has been completed (see TCHK$). One way to check for operation completion is to call TR$ again with a different buffer as its parameter. Return from the second call will be made as soon as the first operation is completed. This is the mechanism via which multiple buffering is normally achieved. Note that tape motion will not cease if TR$ is called within five milliseconds of the end of the previous record.

If parity problems arise, TR$ tries up to 5 times to read the tape before giving a parity failure trap. Other traps given are end of tape and end of file. If an end of file trap is given, the tape is positioned before the file marker.

Entry point: 010031

Parameters:
- B = deck number
- HL => data storage location

Exit conditions: none

Traps:
- D Parity failure occurred.
- E End of tape encountered.
- F End of file encountered.

53.7 TREAD$ - TR$ and Wait for the Last Character

TREAD$ performs the TR$ function and then waits for the last character to be read from the tape. This routine should be used when multiple buffering is not being performed since it relieves the user from having to explicitly wait for the last character to be read.

Entry point: 010034

Parameters: same as for TR$

Exit conditions: none

Traps: same as for TR$

53.8 TW$ - Write a Numeric CTOS Record

TW$ writes the specified memory locations in a record of standard CTOS numeric format. It uses (for parity generation) the three locations preceeding the memory location specified which contains the number of bytes to be written and is followed by that number of data bytes.

TW$ returns as soon as the write operation is started. The user must be careful not to change any of the memory locations given as parameters before the last byte has been transferred. This can be achieved by either calling TCHK$ and waiting for completion status or calling TW$ with the next buffer if multiple buffering is being used. Note that tape motion will not cease if TW$ is called before the middle of the IRG is reached from the previous write (140 milliseconds after the last character is written when using a 7.5 ips deck).

Entry point: 010037

Parameters: same as for TR$

Exit conditions: none

Traps:
- E End of tape encountered.
- Z Premature deck ready status occurred.

53.9 TWRIT$ - TW$ and Wait for the Last Character

TWRIT$ executes the TW$ routine and then waits for the last byte to be written on the tape. This routine should be used when multiple buffering is not being performed since it relieves the user from having to explicitly wait for the last byte to be written.

Entry point: 010042

Parameters: same as for TR$

Exit conditions: none

Traps: same as for TW$

53.10 TFMR$ - Read the Next File Marker

TFMR$ reads the tape until a file marker record is found. A trap occurs if a record is encountered that is neither a file marker nor a CTOS numeric data record.

Entry point: 010045

Parameters: B = deck number

Exit conditions: C = PFN of file marker
Tape positioned after marker record.

Traps: D Unrecognizable record found.
E End of tape encountered.

53.11 TFMW$ - Write a File Marker Record

TFMW$ writes a file marker record that contains the number specified.

Entry point: 010050

Parameters: B = deck number
C = PFN to be written

Exit conditions: none

Traps: E End of tape encountered.
Z Premature deck ready status occurred.

53.12 TTRAP$ - Set an Error Condition Trap

TTRAP$ allows the user to trap the various errors associated with cassette I/O. If the trap is not set and an error occurs, an error message of the form

*** ERROR X ON DECK Y ***

will be displayed, where X is one of the letters shown below and Y is a 1 for the rear deck and a 2 for the front deck. The trap is specified by a number according to the following table:

3 - D - Parity failure occurred.
4 - E - End of tape encountered.
5 - F - End of file encountered.
6 - G - File could not be found.

In addition, error Z (which cannot be trapped) indicates that the deck ready status bit came true while a record was being written. This status implies that the write routine fell behind in writing characters and most probably indicates that the foreground interrupt handling was disrupted in some fashion (interrupts were disabled too long or an interrupt driven routine was running which imposed too much overhead). It may also be caused by the tape being write protected (left rear tab punched out).

Traps are cleared by setting their addresses to zero. When the event which causes a trap occurs, that trap is cleared and control passed to the address indicated with the deck number in the B register (0 for rear and 1 for front deck).

Entry point: 010053

Parameters: C = trap number (above)
DE= trap address (0 clears trap)

Exit conditions: none

Traps: none

53.13 TWAIT$ - Wait for I/O Completion

TWAIT$ waits for any tape operation active to complete. This does not mean that physical motion has stopped since TR$ and TW$ indicate I/O completion when the last character has been transferred. It does mean that all data is free to be processed by the user. TWAIT$ also executes any traps pending upon the completion status being set.

Entry point: 010056

Parameters: none

Exit conditions: B, C, D, and E registers preserved

Traps: Any trap pending will be executed.

53.14 TCHK$ - Get I/O Status

TCHK$ sets the tape demand flag in the Carry condition flag and loads the tape handling status in the A register. The handling status codes are as follows:

000 - PBOF in progress
002 - PEOF in progress
004 - Rewind in progress
006 - Record read in progress
010 - Backspace in progress
012 - File mark read in progress
014 - Record write in progress

377 - Normal completion
206 - Parity error
210 - End of tape
212 - End of file
214 - File not found
262 - Premature deck ready status

Normal use of the cassette routines will not require the user to deal with these status codes or even use the TCHK$ routine. They are provided here to facilitate understanding the listing of the routines.

Entry point: 010061

Parameters: none

Exit conditions: Carry condition = demand flag
A = status code (above)

Traps: none

CHAPTER 54. COMMAND INTERPRETER ROUTINES

This section deals with a series of routines within the command interpreter. Note that these routines are only available for use if the user program does not overlay the command interpreter, which resides in locations 013400-016777.

The first four of these entry points are more like "exit points", since they are places in the DOS to which users may return instead of EXIT$. The primary advantage to using them instead of EXIT$ is that none of these four entry points results in the DOS being reloaded, a process which takes significant time. Since these entry points do not reload the DOS, programs which exit through CMDINT, DOS$, CMDAGN, or NXTCMD must not have overstored any part of the DOS; that is they should reside only in locations 017000 upwards. Also, these "exit points" do not clear any traps that the user may have set; therefore the user should clear any traps he has set before exiting in this manner. If this is not done, the system will most likely go astray upon the first subsequent occurrence of a trapped situation.

Most of the other routines documented in this section are routines which are used by one or more of the DOS command programs supplied either on the DOS Generation or DOS Utilities tapes. Since these routines are in the command interpreter's entry point table and are used by some of the DOS commands, they are documented here primarily for the sake of completeness.

54.1 CMDINT - Return & Scan MCR$ line

CMDINT closes files 1-3 if necessary and processes MCR$ just as it would a command line entered by an operator at the keyboard. (This results in executing the program indicated by the command line.)

Entry point: 01165

Parameters: MCR$ (an 80 byte area of memory starting at 01400) should contain a string resembling a command line terminated with a 015.

Exit conditions: Does not return

54.2 DOS$ - Return & Display Sign On

DOS$ first loads the RAM screen, if available, with the character set contained in SYSTEM6/SYS (or CHARSET/SYS if it exists). Once the RAM display has been loaded, it is not reloaded until either another bootstrap from cassette, or the appropriate DOS function is invoked by a DOS program. DOS$ then causes any program which has been AUTOed to be executed. If no programs are set for auto-execution, the DOS sign-on is displayed, files 1-3 are closed if necessary, and the familiar "READY" message is displayed. Any traps set by the user program (via TRAP$) are not cleared unless the DOS is reloaded. This implies that if a user program sets any of the traps and wishes to return via DOS$, NXTCMD, or CMDAGN, it must first clear any traps it has set to prevent the DOS from going astray. DOS$ is the normal starting point of the DOS when a bootstrap operation or a jump to BOOT$, EXIT$, or ERROR$ occurs.

Entry point: 013400

Parameters: none

Exit conditions: Does not return

54.3 NXTCMD - Return and Dislay "READY"

NXTCMD causes files 1-3 to be closed and displays the familiar DOS "READY" message.

Entry point: 013403

Parameters: none

Exit conditions: Does not return

54.4 CMDAGN - Return & Give Message

CMDAGN causes files 1-3 to be closed and displays a user-supplied message before returning to the command interpreter. CMDAGN causes termination of an active chain and is intended as an error exit.

Entry point: 013406

Parameters: HL => DSPLY$-format string
DE unused; string should position cursor

Exit conditions: Does not return.
DOS CHAIN facility aborts if active.

54.5 GETSYM - Get Next Symbol

GETSYM causes the symbol pointed to by INPTR to be scanned off and stored in an 8-byte field called SYMBOL located at 013472. The symbol (leading spaces are ignored) must contain only upper case alphabetic or numeric characters. The first illegal character encountered terminates the scan; the illegal, terminating character is stored for the user' s inspection (at SYMBOL+8) and SYMBOL is padded on the right with spaces if necessary. If the symbol is longer than eight characters, the first eight only are used; remaining characters, through the terminator, are scanned but not stored. (The terminator is stored at SYMBOL+8 in any case.) On exit, INPTR points to the byte following the terminator, unless the terminator was an 015 or a semicolon, in which case INPTR points to the terminator.

Entry point: 013411

Parameters: INPTR, INPTR+1 => beginning of symbol

Exit conditions: SYMBOL = 8-byte symbol as described above
A, SYMBOL+8 = terminator character
INPTR, INPTR+1 => byte after symbol terminator
(except as noted above)
all other registers indeterminate

54.6 GETCH - Get the Next Character

GETCH obtains the next character pointed to by INPTR and returns it in A. On exit, if zero is true, A = 015 or a semicolon, and INPTR is not incremented (INPTR is never bumped past an 015 or a semicolon); if zero is false, A is not an 015 or a semicolon and INPTR is incremented.

Entry point: 013414

Parameters: INPTR, INPTR+1 = address of byte (see above)

Exit conditions: A = character
ZERO TRUE/FALSE as described above
INPTR, INPTR+1 = HL+1 if zero false or
HL if zero true
C,D,E unchanged

B indeterminate

54.7 GETAEN - Get Auto-Execute Physical File Number

GETAEN returns the physical file number of the file (on the logical drive specified in C) which is set to be auto-executed by the DOS.

Entry point: 013417

Parameters: C = Logical Drive

Exit conditions: Carry true if I/O error reading the CAT
Carry false: A = auto-execute PFN (0=none)
Zero true if a-e PFN not set
Zero false if A is valid a-e PFN
all other registers indeterminate

54.8 PUTAEN - Set or Clear a File to be Auto-Executed

PUTAEN either sets or clears the auto-execute PFN stored in the CAT on the disk in the logical drive specified in C. The change becomes effective upon the next time DOS is entered at DOS$, either by depressing the RESTART key, the auto-restart tab being punched out of the rear cassette and the processor halted, or jumping to EXIT$, ERROR$, BOOT$, or DOS$.

Entry point: 013422

Parameters: A = PFN to be auto-executed (0 to clear)
C = Logical Drive

Exit conditions: all registers indeterminate
Carry true if I/O error updating CAT

54.9 GETLFB - Open the User-Specified Data File

GETLFB opens logical file specified in B using the file name, extension, and drive select code stored in the indicated LFT entry, in the normalized form described in the section on the Command Interpreter. The extension, if blank, is assumed to be "ABS". The logical drive specification field is ignored, since the drive select code field is used instead. If an error occurs, Carry is true on return and HL points to a DSPLY$-format string complete with cursor positioning bytes and one of the following

messages:

```
NAME REQUIRED.  First byte of name field is blank.
INVALID DEVICE.  Drive specification invalid; select
                   code = 0376.
NO SUCH NAME.  File not found; the file must exist.
```

Each of the above messages is preceeded by control bytes: 011,0,013,11,023 and followed by an 015. If carry is false upon return, the file named has been successfully opened as the requested logical file number.

Entry point: 013425

Parameters: B = LFN
Other parameters in LFT specified by LFN; see above.

Exit conditions: Carry false: file successfully opened
all registers indeterminate
Carry true: OPEN failed
HL => message

54.10 PUTCHX - Store the Character in "A"

PUTCHX stores the A register at the memory location pointed to by HL, increments HL, and decrements a byte counter maintained in E.

Entry point: 013433

Parameters: HL = address where A is to be stored
A = byte to be stored at HL
E = count to be decremented

Exit conditions: B,C,D unchanged
E = entry value - 1
HL = entry value + 1

54.11 PUTCH - Alternate Version of PUTCHX

PUTCH is like PUTCHX except it starts by setting the most significant bit of A to zero. If A then contains a space (040) PUTCH immediately returns with zero true, in which case A is not stored, HL is not incremented, and E is not decremented.

Entry point: 013430

Parameters: same as PUTCHX

Exit conditions: same as PUTCHX except as described above

54.12 PUTNAM - Format a Filename from Directory

PUTNAM is a routine which extracts a name, extension and physical file number for a directory entry and puts them into a place in the command interpreter called "NAME" (located at 013513; the field is 19 bytes long and followed by an 03.) Since this routine is used by the CAT command, the format of the names produced by PUTNAM should be familiar to all DOS users.

Note that on entry, only the most significant 4 bits of C are used, and that CURLOC (location 013463) is to contain the two-byte PDA of the directory sector (LSB, MSB).

Entry point: 013436

Parameters: directory sector in the disk buffer
B = LFN indicating which buffer
C = PFN of entry being extracted
CURLOC = PDA of directory sector

Exit conditions: CURLOC unchanged
disk buffer unchanged
B unchanged
all other registers indeterminate
ZERO TRUE: file does not exist

54.13 MOVSYM - Obtain the Symbol Scanned by GETSYM

MOVSYM moves the eight-byte SYMBOL described in the section on GETSYM into the eight-byte area pointed to by DE.

Entry point: 013441

Parameters: DE = address of user's eight-byte area

Exit conditions: B unchanged.
all other registers indeterminate

54.14 GETDBA - Obtain Disk Controller Buffer Address

GETDBA extracts the current disk buffer address, in a format acceptable to GETR$, from one of the four LFT entries. It does this by getting the BUFADR from the specified LFT entry and subtracting three from it. On return, H is the address MSB pointing into the command interpreter data area.

Entry point: 013444

Parameters: B = LFN (0,16,32,48)

Exit conditions: A = BUFADR as described above
H as described above
B,C,D,E unchanged

54.15 SCANFS - Scan Off File Specification

SCANFS scans a file specification of the form <filename>/<ext>:<drv> pointed to by HL into a 16 byte area pointed to by DE. The area pointed to by DE is treated as an LFT entry, that is, the first byte is a drive select code (0376 meaning invalid drive spec, 0377 meaning unspecified drive spec, or the binary drive number), the second byte is 0377 indicating the file is closed, bytes 3 thru 10 are the file name (blank if not given), bytes 11 thru 13 are the extension (blank if not given), and bytes 14 thru 16 are the normalized drive spec (blank if not given). The scanned drive spec may be 1 to 8 characters long and must be in volid (:<volid>) or drive number (:Dn or :DRn) form. For a drive number the drive entry must be 2-7 characters long, the first character must be "D", the second may be "R", and the remaining must be digits. Therefore ":D0" and ":DR00014" are both legal representations. The normalized represention consists of a "D" followed by "R" and the single digit given or "D"

followed by the two digits given; for instance, the above examples in normalized form would be "DR0" and "D14" respectively. Scanning a volid results in the correct drive number being stored in the normalized drive spec field. The scan is terminated by any non-alphanumeric character other than ":" or "/".

Entry Point: 013447

Parameters: DE => "LFT TABLE" entry
HL => string to be scanned

Exit Conditions: DE => byte following "LFT TABLE" entry
HL => byte after terminator (unless 015 or ";" in which case it points to terminator)

54.16 TCWAIT - Test controller memory & wait

TCWAIT is the point in the COMMAND INTERPRETER where it loops testing the disk controller buffer memory while waiting for a command to be keyed in. It is only to be used by the CHAIN command to trap programs returning to DOS.

Entry Point: 013452

Parameters: none

Exit Condition: Does not return.

CHAPTER 55. USER SUPPORTED INPUT/OUTPUT

When the user desires to use I/O devices other than the keyboard, display, disk, or cassettes, he will use a routine that is not part of the operating system. Many of these devices (for instance, the communications channel) will be serviced by foreground processes which run with interrupts disabled. However, if the user does access an I/O device from a background process, he must realize that as long as interrupts are enabled, some other device can be addressed by a foreground routine. For this reason the user must disable interrupts between the time he addresses his device and the time he uses it. To reduce the amount of foreground processing real time jitter (discussed earlier) as much as possible, the aim in writing background I/O routines should be to minimize the amount of time that interrupts are disabled. This implies that devices accessed from background programs must be addressed every time they are used. For example:

```
GETBYT  EI              Enable interrupts in case
        LA    DEVADR       looping
        DI              Disable interrupts
        EX    ADR       Address the device
        IN              Get the device status
        ND    2         Check for required bits
        JTZ   GETBYT    Wait if not set
        EX    DATA      Else get the byte
        EI              Enable interrupts after
        IN                 the data input
        RET
```

Note that a little cheating on time was done in the interest of program length. Since the INPUT in DATA mode was done without enabling interrupts, re-disabling them and re-addressing the device was not necessary. One should be judicious in the trade off employed in exercising this freedom.

The user must not do I/O to the disk controller from foreground-driven routines or results can be unpredictable. The DOS disk drivers allow user foreground routines to receive control in the midst of a disk I/O operation, under the assumption that the foreground routine will not do anything to the disk controller which would confuse it.

CHAPTER 56. ERROR MESSAGES

56.1 System Error Messages

The following error messages are produced by the DOS system routines and may appear during the execution of almost any program.

PARITY FAILURE DURING READ
: A parity fault occurred while a disk data record was being read.

PARITY FAILURE DURING WRITE
: A parity fault occurred while a disk data record was being written.

RECORD FORMAT ERROR
: The physical file number or logical record number in the record read did not match the values contained in the logical file table.

RECORD NUMBER OUT OF RANGE
: The record accessed had a logical record number less than zero or, during reads, was outside the physical space allocated to the file.

WRITE PROTECT VIOLATION
: An attempt was made to write on a file that had its write protection bit set.

DELETE PROTECT VIOLATION
: An attempt was made to delete a file that had either its write or delete protection bit set.

FILE SPACE FULL
: An attempt was made to allocate space when either the disk was physically full or no more segment descriptor slots were available in the RIB for the given file.

DRIVE OFF LINE
The drive went off-line after a file on it was opened.

LOGICAL FILE NOT OPEN
An attempt was made to use an entry in the logical file table that was not opened for use with some file.

INVALID LOGICAL FILE NUMBER
A routine was called with the logical file number parameter not 0, 16, 32 or 48.

INVALID DRIVE NUMBER
A routine was called with the drive number not zero through the defined drive number limit (or 0377, if allowed).

INVALID TRAP NUMBER
The TRAP$ routine was called with a trap number not between zero and seven.

FAILURE IN SYSTEM DATA
An unrecoverable parity error occurred while the system was dealing with one of the disk tables or a retrieval information block, or a RIB with incorrect format was accessed.

INVALID PHYSICAL FILE NUMBER
A physical file number reserved for the system was illegally referenced.

INTERNAL SYSTEM ERROR
The error message routine was parameterized with an invalid error message number.

ERROR X ON DECK Y
A cassette routine error has occurred. The X indicates the type of error according to the following table:

D - parity error
E - end of tape
F - end of file
G - unfindable file
Z - write failure

56.2 Utility Program Error Messages

The following messages (listed in alphabetic order) are produced by the indicated DOS utility. For a description of the error condition and possible corrective action see the individual chapter for the utility in use.

'7' OPTION IS ONLY VALID WHEN COPYING SYSTEM7/SYS
    COPY

* DRIVE OFF LINE
    SUR

* DRIVE OFF LINE OR FULL *
    SUR

* DRIVE OFF-LINE *
    AUTO

* DRIVE OFFLINE *
    FILES

* INVALID EXTENSION *
    SUR

* MAY NOT BE DELETED *
    SUR

* MAY NOT BE RENAMED *
    SUR

* PARITY ERROR *
    FILES

* SYSTEM DATA FAILURE *
    AUTO,CAT,MANUAL,NAME

* SYSTEM DATA FAILURE ON DRIVE x *
    SUR

* SYSTEM DRIVE OFF LINE *
    MANUAL

** BAD DATA IN THE SOURCE FILE LINE DIGITS **
    BLOKEDIT

** BAD FILE SPECIFICATION **
    BLOKEDIT

** BAD LINE NUMBER SPECIFICATION **
    BLOKEDIT

** FORMAT OR RANGE ERROR ON SOURCE FILE **
    BLOKEDIT

** LINE NUMBER ZERO IS NOT VALID **
    BLOKEDIT

** NO VALID SOURCE FILE FOR TRANSFER **
    BLOKEDIT

** SOURCE FILE NOT FOUND **
    BLOKEDIT

** SOURCE FILE WENT TO EOF **
    BLOKEDIT

** START LINE NO. > END LINE NO. **
    BLOKEDIT

```
*** BAD OPTION FOLLOWING ; ON COMMAND LINE - IGNORED ***
     BLOKEDIT
*** COMMAND AND NEW FILE NAMES CAN NOT BE IDENTICAL ***
     BLOKEDIT
*** COMMAND FILE DRIVE IS INVALID ***
     BLOKEDIT
*** COMMAND FILE NOT FOUND ***
     BLOKEDIT
*** DUPLICATE NAME.  NOT COPIED. ***
     BACKUP
*** INCORRECTLY FORMATTED DISK RECORD ***
     MOUT
*** NEW FILE AND SOURCE FILES CAN NOT BE IDENTICAL ***
     BLOKEDIT
*** NEW FILE DRIVE IS INVALID ***
     BLOKEDIT
*** NEW FILE NAME IN USE ***
     BLOKEDIT
*** NEW FILE NAME IN USE, OVERWRITE IT ?  ANSWER WITH YES OR NO
     BLOKEDIT
*** NEW FILE NAME IS REQUIRED ***
     BLOKEDIT
*** NOT ALL FILES WOULD FIT ON THE CASSETTE ***
     MOUT
*** THE FILE IS TOO LARGE TO FIT ON THE CASSETTE ***
     MOUT
*** UNABLE TO OPEN INPUT FILE! ***
     BACKUP
***END OF TAPE***
     MIN
***MISMATCH***
     MOUT
***PARITY ERROR - FILE WILL BE DELETED***
     MIN
***PARITY ERROR - RECORD MODIFIED***
     MIN
***WRITE PROTECTED***
     MIN
*END OF TAPE WHILE WRITING DIRECTORY*
     MOUT
*FILE CONTAINS NON-OBJECT RECORD*
     MOUT
*FILE FORMAT ERROR*
     REFORMAT
*FILE NOT OPEN*
     REFORMAT
*INCORRECTLY FORMATTED SOURCE RECORD*
     MOUT
```

```
- ALREADY IN DRIVE x'X SUBDIRECTORY.
     SUR
- DRIVE x HAS NO SUBDIRECTORY.
     SUR
- DRIVE x IS A REMOTE ARC VOLUME.
     SUR
- DRIVE x'S SUBDIRECTORY IS FULL.
     SUR
- NOT IN DRIVE x'S SUBDIREECTORY.
     SUR
BAD CLUSTER ALLOCATION TABLE! USE DSKCHECK TO FIX IT.
     BACKUP
BAD DATE
     MOUT
BAD DEVICE SPECIFICATION.
     CHAIN
BAD DEVICE.
     EDIT
BAD DOS FUNCTION
     FREE
BAD DRIVE
     MOUT
BAD DRIVE SPECIFICATION
     FREE
BAD EXTENSION (/XTX) FOR SCRATCH.
     EDIT
BAD FILE FORMAT!
     EDIT
BAD LOADER
     MOUT
BAD OPTION PARAMETER
     DUMP,EDIT,MIN
BAD OPTION PARAMETER.  MOUT DISCONTINUED.
     MOUT
BAD SECOND FILE SPEC!  SHOULD BE "Ln" OR "Rn"!
     LIST
BAD TAPE DIRECTORY
     MOUT
BLOCK TOO SHORT ON VERIFY
     UBOOT
BLOCKING FACTOR CONTAINS INVALID NON-NUMERIC DIGITS.
     REFORMAT
BLOCKING FACTOR REQUIRED BUT MISSING OR ZERO FOUND.
     REFORMAT
BOOT GENERATION ABORTED - TRY AGAIN?
     UBOOT
BOOTED DRIVE IS ILLEGAL OUTPUT DURING REORG.
     BACKUP
```

```
BOTH SOURCE AND SCRATCH FILES CANNOT BE SAME.
     EDIT
CAN'T FIND FILE xxxxxxxx/xxx MEMBER xxxxxxxx
     FIXAPPLY
CASSETTE NO LONGER IN PLACE
     UBOOT
CASSETTE WRITE PROTECTED
     UBOOT
CHAIN OVERLAY MISSING.
     CHAIN
CHAINING ABORTED - ABTIF - ERROR BIT ON.
     CHAIN
CHAINING ALREADY ACTIVE.
     CHAIN
COMMAND LINE ERROR: 015 MISSING
     REFORMAT
CONDITION SPECIFICATION ERROR.
     CHAIN
CONFLICTING OPTIONS SPECIFIED.
     FILES
CORRECT FORMAT IS:  PUTVOLID <VOL-ID><:DRIVE>;<OWNER-ID>
     PUTVOLID
CRC ERROR ON DRn
     PUTVOLID
CYLINDER ZERO OF BACKUP DISK IS UNUSABLE!
     BACKUP
DECK ERROR (DECK READY DURING WRITE)
     UBOOT
DIRECTORY FULL
     MOUT
DISK EOF BEFORE TAPE EOF.
     MOUT
DISK FILE CONTAINS NON-OBJECT RECORD.
     MOUT
DISK FILE CONTAINS NON-TEXT RECORD.
     MOUT
DISK OFFLINE
     DUMP
DISK SECTOR CRCC ERROR
     DUMP
DISK SECTOR FORMAT ERROR
     DUMP
DOS FUNCTION 15 ERROR
     FILES
DRIVE OFF LINE
     DSKCHECK,KILL
DRIVE xx OFFLINE!
     BACKUP
```

DRn OFF-LINE
     PUTVOLID
DUPLICATE KEY:
     INDEX
ERROR IN DOS FUNCTION. DUMP ABORTED.
     DUMP
ERROR IS DOS FUNCTION
     FILES
EXCESS FILE SPACE NOT DEALLOCATED; OUTPUT FILE IS DELETE PROTECTED
     REFORMAT
FAULTY DEFINITION FILE.
     EDIT
FILE ALREADY WAS IN SPECIFIED FORMAT.
     REFORMAT
FILE EXHAUSTED BEFORE LINE FOUND. FILE IS nnnn LINES LONG.
     LIST
FILE FOLLOWING LOADER NOT OBJECT
     MOUT
FILE INTEGRITY ERROR ON INPUT!
     FIXAPPLY
FILE IS MISSING FROM UTILITY/REL, OR IS UNLOADABLE
     LIST
FILE NAME MISSING
     CHANGE
FILE NOT FOUND
     DUMP,EDIT,MIN,MOUT
FILE NOT FOUND. CREATE IT?
     EDIT
FILE TWO AND THREE MUST BE DIFFERENT.
     APP,SAPP
FILE x CONTAINS A NON-OBJECT RECORD.
     APP
FILE(S) NOT FOUND
     FILES
FORMAT - NEXT RECORD NUMBER:
     LIST
FORMAT ERROR IN INPUT FILE!
     LIST
HADCOPY ONLY IF LIMITED OUTPUT
     SORT
HARDCOPY HEADING STRING MISSING
     SORT
HOW DO YOU EXPECT TO FIT THAT MANY RECORDS IN A 256-BYTE SECTOR?
     REFORMAT
I CAN'T RUN IN THE REMOTE PARTITION
     EDIT
I CAN'T WRITE A NULL TAPE!
     MOUT

```
ILLEGAL EOF ON INPUT!
     FIXAPPLY
ILLEGAL HEADER KEY EVALUATION
     SORT
ILLEGAL HEADER SPECIFICATION
     SORT
ILLEGAL SORT KEY SPECIFICATION
     SORT
ILLEGAL, CONFLICTING OR DUPLICATE OPTION.
     REFORMAT
INDEX FILE DOESN'T POINT TO ITSELF!
     LIST
INDEX TERMINATED WITH DUPLICATE KEYS
     INDEX
INFILE NAME MISSING
     INDEX
INPUT AND OUTPUT DRIVES MUST BE DIFFERENT
     BACKUP
INPUT AND OUTPUT FILES CANNOT BE THE SAME!
     LIST
INPUT DISK LOCKOUT CAT UNREADABLE.  REORGANIZATION REQUIRED.
     BACKUP
INPUT FILE CONTAINS BAD DATA!
     DECODE
INPUT FILE DOES NOT EXIST!
     DECODE,ENCODE
INPUT FILE IS EMPTY!
     REFORMAT
INPUT FILE MISSING OR NOT SPECIFIED!
     FIXAPPLY
INPUT FILE MUST BE SPECIFIED
     DECODE,ENCODE
INPUT FILE MUST EXIST IN "ONE-PASS".
     EDIT
INPUT FILE NOT FOUND
     LIST,SORT
INPUT FILE REQUIRED
     SORT
INTERNAL ERROR
     EDIT
INTERNAL ERROR -- GET SYSTEMS HELP !!!
     SORT
INVALID DEVICE
     CAT,INDEX,SAPP
INVALID DEVICE SPECIFICATION
     LIST
INVALID DRIVE
     APP,BACKUP,DSKCHECK,FILES,KILL,MIN,PUTIPL,SORT
```

```
INVALID DRIVE SPECIFICATION
     BUILD
INVALID END-OF-FILE AT LRN nnnnn.
     LIST
INVALID END-OF-FILE MARK AT LRN nnn
     SORT
INVALID FILE SPECIFICATION
     BUILD
INVALID FORMAT FOR PFN REPLY.
     KILL
INVALID HADCOPY HEADING ADDRESS
     SORT
INVALID LIMITATION SPECIFICATION
     SORT
INVALID LIMITATION STRING ADDRESS
     SORT
INVALID OBJECT FORMAT!
     FIXAPPLY
INVALID OPTION PARAMETER
     DSKCHECK
INVALID OPTIONS SPECIFIED.  VALID ONES ARE:
     REFORMAT
INVALID PHYSICAL ADDRESS
     DUMP
INVALID PROTECTION SPECIFICATION.
     CHANGE
INVALID TAPE FORMAT
     MIN,MOUT
INVALID TEXT CHARACTER AT LRN nnn
     SORT
INVALID USER EXIT ADDRESS
     SORT
ISAM AND LINE/RECORD COUNT INCOMPATIBLE
     LIST
KEY FILE SPECIFICATION ERROR
     SORT
LFT ENTRIES 1->3 NOT CLOSED WHEN SORT ENTERED
     SORT
LIMITATION STRING MISSING
     SORT
LINE OVERFLOW DURING VALUE SUBSTITUTION.
     CHAIN
LIST TERMINATED.
     LIST
LOCAL IS MISSING FROM UTILITY/REL OR UNLOADABLE
     FILES
LOCAL IS MISSING FROM UTILITY/REL, OR IS UNLOADABLE
     LIST
```

```
LOGICAL RECORD LENGTH REQUIRED BUT MISSING OR ZERO FOUND.
     REFORMAT
LOGICAL RECORD LENGTH MUST BE < 65535 BYTES.
     REFORMAT
LONG KEY ENCOUNTERED AND TRUNCATED
     INDEX
MISSING END-OF-PHYSICAL RECORD AT LRN nnnn.
     LIST
MISSING EOS AT LRN nnn
     SORT
MISSING OWNER-ID.
     PUTVOLID
MULTIPLE PRINTERS SELECTORS
     DSKCHECK
NAME IN USE.
     NAME
NAME NOT FOUND.
     APP,SAPP
NAME REQUIRED
     APP,CHAIN,EDIT,MOUT,NAME,SAPP
NO AUTOKEY LINE CONFIGURED.
     AUTOKEY
NO CASSETTE IN SELECTED DECK.
     REWIND
NO SUCH NAME
     CHAIN,INDEX,KILL
NO SUCH OVERLAY LIBRARY.
     CAT
NO SUCH SUBDIRECTORY
     NAME
NO!
     KILL
NO!  YOU CAN ONLY "AUTO" FILES THAT ARE ON BOOTED DRIVE.
     AUTO
NO!  YOU CANNOT "AUTO" SYSTEM FILES.
     AUTO
NO! THAT FILE IS PROTECTED.
     NAME
NON-SEQUENTIAL FILE MARK
     MOUT
NOT DIRECTORY TAPE
     MOUT
NOT LGO TAPE
     MOUT
NOT WHILE ARC IS RUNNING
     BACKUP,PUTIPL,PUTVOLID,UBOOT
NOT WHILE PS IS RUNNING
     BACKUP,PUTIPL,PUTVOLID,UBOOT
```

```
NULL FILE
     MOUT
NULL INDEX FILE CREATED
     INDEX
NULL LIMITATION STRING
     SORT
NUMBER NOT OCTAL
     MIN
NUMBER OF LINES PER PAGE MUST BE 1-255!
     LIST
OLD/NEW BYTE MISMATCH!
     FIXAPPLY
OPTION FIELD ERROR
     SORT
OPTION SPECIFICATION DUPLICATION
     SORT
OPTION SPECIFICATION ERROR.
     CHAIN
OUTPUT DISK LOCKOUT CAT UNREADABLE.  REORGANIZATION REQUIRED
     BACKUP
OUTPUT DRIVE NUMBER REQUIRED.
     PUTVOLID
OUTPUT FILE IS WRITE PROTECTED AND CANNOT BE WRITTEN INTO OR
SHORTENED.
     REFORMAT
OUTPUT FILE NOT FOUND ON DRIVE x.
     REFORMAT
OUTPUT FILE OVERFLOW
     SORT
OUTPUT FILE SAME AS INPUT
     SORT
OUTPUT FILE WOULD DESTROY INPUT FILE!
     DECODE,ENCODE
OUTPUT FILE WRITE PROTECTED.
     EDIT
PARAMETER ERROR MULTIPLE PRINT DEVICES REQUESTED.
     LIST
PARAMETER ERROR, ALLOWED PARAMETERS ARE D,L,S,P,Q,X,Nn,F AND I
     LIST
PARITY ERROR ON DRIVE n.  PDA:  nnnn,nnnn
     BACKUP
PARITY ERROR ON READ.  LRN: nnnnn
     BACKUP,COPY
PARITY ERROR ON WRITE.  LRN: nnnnn
     BACKUP,COPY
PHASE 1 ERRORS; ABORTED
     FIXAPPLY
PHASE 2 INTERNAL ERROR; ABORTED
```

```
     FIXAPPLY
PHASE 3 INTERNAL ERROR; ABORTED
     FIXAPPLY
PLEASE DOSGEN YOU OUTPUT DISK FIRST
     BACKUP
PRINT DRIVER MEMBER OF UTILITY/REL IS WRONG VERSION, I NEED
VERSION A
     DSKCHECK
PRINT MODULE MISSING OR NOT LOADABLE NO PRINT WILL OCCUR
     DSKCHECK
PRINTER MODULE MISSING FROM UTILITY/REL.
     DUMP
PROGRAM ERROR; EXCESS FILE SPACE NOT DEALLOCATED TO PREVENT
POSSIBLE LOSS OF
DATA.
     REFORMAT
PROGRAM NOT LOADABLE
     DSKCHECK
PROTECTION UNCHANGED.
     CHANGE
RANGE - NEXT RECORED NUMBER:
     LIST
RANGE ERROR IN INPUT FILE!
     LIST
RECORD COUNT ERROR!
     FIXAPPLY
RECORD FORMAT ERROR
     FIX
REFORMAT UNLOADABLE!
     INDEX
RELOCATABLE MEMBER MISSING OR UNLOADABLE
     DSKCHECK,FIXAPPLY
SCREEN IS MISSING FROM UTILITY/REL OR UNLOADABLE
     FILES
SCREEN IS MISSING FROM UTILITY/REL, OR IS UNLOADABLE
     LIST
SEC2200 IS MISSING FROM UTILITY/REL OR UNLOADABLE
     FILES
SEC5500 IS MISSING FROM UTILITY/REL OR UNLOADABLE
     FILES
SECPS IS MISSING FROM UTILITY/REL OR UNLOADABLE
     FILES
SECTOR IN/OUT MEMBER OF UTILITY/REL IS WRONG VERSION, I NEED
VERSION A
     DSKCHECK
SECTOR OUT OF RANGE
     DUMP
SEQUENCE FILE FORMAT ERROR n
```

```
     SORT
SEQUENCE FILE NAME REQUIRED
     SORT
SEQUENCE FILE NOT FOUND
     SORT
SERVO IS MISSING FROM UTILITY/REL OR UNLOADABLE
     FILES
SERVO IS MISSING FROM UTILITY/REL, OR IS UNLOADABLE
     LIST
SORT KEY TOO LONG
     SORT
SORT OVERLAY MISSING
     SORT
SORT UNLOADABLE!
     INDEX
SPECIFIED OUTPUT FILE FORMAT ENLARGES PRESENT INPUT FILE.  F
     REFORMAT
SUBDIRECTORY NOT FOUND.
     FILES
SYMBOL TABLE OVERFLOW
     CHAIN
SYSTEM7/SYS MISSING!
     INDEX,KILL
SYSTEMx/SYS IS MISSING
     BACKUP
TAG DEFINED MORE THAN ONCE.
     CHAIN
TAG VALUE NOT TERMINATED.
     CHAIN
TAPE EOF BEFORE DISK EOF.
     MOUT
TAPE FILE MARK READ BEFORE TAPE OBJECT EOF.
     MOUT
TAPE OBJECT EOF NOT FOLLOWED BY TAPE FILE MARK
     MOUT
THAT DRIVE HAS NO SUBDIRECTORY
     NAME
THAT FILE IS NOT IN YOUR SUBDIRECTORY.
     KILL
THAT ISN'T THE RIGHT FORMAT FOR YOUR REPLY.
     KILL
THE CHAINP/SYS FILE HAS BEEN DELETED.
     CHAIN
THE DISKS' LOCKOUT-OUT CYLINDERS DO NOT MATCH.  REORGANIZATION
REQUIRED.
     BACKUP
THE KEYBOARD KEY WAS HIT MIN ABORTED
     MIN
```

```
THIS PROGRAM IS RUNNING FROM IN A CHAIN - THE "F" OPTION HAS BEEN
DEACTIVATED
     DSCHECK
THIS PROGRAM IS RUNNING FROM UNDER ARC - THE "F" OPTION HAS BEEN
DEACTIVATED
     DSKCHECK
THIS PROGRAM IS RUNNING FROM UNDER PS - "F" OPTION HAS BEEN
DEACTIVATED
     DSKCHECK
UNABLE TO LOAD PRINTER MODULE FROM UTILITY/SYS.
     DUMP
UNDEFINED CHAIN OPERATOR.
     CHAIN
UNRECOGNIZABLE TAPE RECORD FORMAT
     MIN
UNRECOGNIZEABLE OPTION CODE.
     FILES
USE 'O' OPTION ON COMMAND LINE TO OVER-WRITE EXISTING OUTPUT FILE
     BLOKEDIT
UTILITY/REL MISSING FROM BOOTED DRIVE!
     FIXAPPLY
UTILITY/REL FILE MISSING
     DSKCHECK
UTILITY/REL IS MISSING FROM BOOT DRIVE
     FILES
UTILITY/REL IS MISSING ON BOOTED DRIVE
     LIST
UTILITY/REL MISSING FROM BOOTED DRIVE.
     DUMP
VALID PARAMTERS ARE 'ON' AND 'OFF'
     ABTONOFF
VERIFICATION UNSUCCESSFUL
     MOUT
VOLUME NAME MISSING.
     PUTVOLID
WRONG DOS!!
     ABTONOFF,APP,AUTO,AUTOKEY,BACKUP,BLOKEDIT,BUILD,CAT,CHAIN,CHA
NGE,
     COPY,DECODE,DSKCHECK,EDIT,ENCODE,FIX,FIXAPPLY,FREE,INDEX,INIT
DISK,
     KILL,MANUAL,MIN,MOUT,NAME,PUTIPL,PUTVOLID,REFORMAT,REWIND,SAP
P,
     SORT,SUR,UBOOT
WRONG PROCESSOR!
     EDIT
YOU CAN'T AUTO THAT FILE !
     AUTO
YOU CAN'T FIX AN OVERLAY LIBRARY!
```

FIX
YOU CANNOT APPEND OVERLAY LIBRARIES.
APP
YOU HAVE ILLEGALLY ENTERED A SPECIFICATION FOR A THIRD FILE
REFORMAT
YOU SPECIFIED AN OUTPUT FILE THAT ENDED UP BEING YOUR INPUT FILE.
TO REFORMAT-IN-PLACE DO NOT SPECIFY ANY OUTPUT FILE.
REFORMAT
YOU SPECIFIED BOTH SEGMENTATION AND TRUNCATION. YOU CAN NOT HAVE
BOTH.
REFORMAT
YOUR BLOCKING FACTOR IS TOO LARGE FOR THE SIZE OF RECORDS YOU
HAVE.
REFORMAT
YOUR DOS FUNCTION 15 IS OBSOLETE
DSKCHECK
YOUR LOGICAL RECORD LENGTH IS TOO SMALL FOR THE SIZE OF RECORDS
YOU HAVE.
REFORMAT

CHAPTER 57. ROUTINE ENTRY POINTS

These entry points are contained in a file called DOS/EPT.

Loader Routines

01000	BOOT$	reload the operating system
01003	RUNX$	load and run a file by number
01006	LOADX$	load a file by number
01047	GETNCH	get the next disk buffer byte
01052	DR$	read a sector into the disk buffer
01055	DW$	write a sector from the disk buffer
01060	DSKWAT	wait for disk ready
01173	DWNV$	DW$ without write verify 2.3 only

Time-critical Scheduling Routines

01033	CS$	change process state
01036	TP$	terminate process
01041	SETI$	initiate foreground process
01044	CLRI$	terminate foreground process

Symbolic File Handling Routines

01063	PREP$	open or create a file
01066	OPEN$	open an existing file
01071	LOAD$	load a file by name
01074	RUN$	load and run a file by name

Logical File Handling Routines

01077	CLOSE$	close a file
01102	CHOP$	delete space in a file
01105	PROTE$	change the protection on a file
01110	POSIT$	position to a record within a file
01113	READ$	read a record into the buffer
01116	WRITE$	write a record from the buffer
01121	GET$	get the next buffer character
01124	GETR$	get an indexed buffer character
01127	PUT$	store into the next buffer position
01132	PUTR$	store into an indexed buffer position
01135	BSP$	backspace one record

Generalized Processing Routines

01011	INCHL	increment HL
01022	DECHL	decrement HL
01140	ERROR$	close all files, exit chain, and reload DOS
01143	BLKTFR	transfer a block of memory
01146	TRAP$	set a disk error condition trap
01151	EXIT$	reload the operating system
01170	WAIT$	DOS wait-a-while "NOP" routine
07400	DOSFNC	DOS function loader

Keyboard and Display Routines

01154	DEBUG$	enter the debugging tool
01157	KEYIN$	obtain a line from the keyboard
01162	DSPLY$	display a line on the screen

Cassette Handling Routines

010000	TPBOF$	position to the beginning of a file
010005	TPEOF$	position to the end of a file
010012	TRW$	physically rewind a cassette
010017	TBSP$	physically backspace one record
010024	TWBLK$	write an unformatted block
010031	TR$	read a numeric CTOS record
010034	TREAD$	TR$ and wait for last character
010037	TW$	write a numeric CTOS record
010042	TWRIT$	TW$ and wait for last character
010045	TFMR$	read the next file marker record
010050	TFMW$	write a file marker record
010053	TTRAP$	set a cassette error trap
010056	TWAIT$	wait for I/O completion
010061	TCHK$	get I/O status

Command Interpreter Utility Routines

01165	CMDINT	return to command interpreter & scan MCR$ line
013400	DOS$	return to command interpreter & display sign on
013403	NXTCMD	return to command interpreter & say "READY"
013406	CMDAGN	return to command interpreter & give message
013411	GETSYM	get the next symbol from MCR$
013414	GETCH	get the next character from MCR$
013417	GETAEN	get the auto execute PFN
013422	PUTAEN	set the auto execute DFN
013425	GETLFB	open the user-specified file (LFN in B)
013430	PUTCH	store the nonblank character in the A register
013433	PUTCHX	store the character in the A register
013436	PUTNAM	format a filename from a directory block

013441	MOVSYM	obtain the symbol scanned off by GETSYM
013444	GETDBA	obtain the disk controller buffer address
013447	SCANFS	scan off a file specification
013452	TCWAIT	test controller memory and wait for command
013455	INPTR	internal byte pointer for GETCH and GETSYM

Internal DOS Equivalences

00004	DOSPFN	PFN for use by DR$ and DW$
00005	DOSPDN	PDN for use by DR$ and DW$
00026	DOSPTR	BUFPTR used by GETNCH
00027	SDFLAG	Sub-directory existence flag
00030	SDNR	Subdirectory number table (one per drive)
00053	BOOTDRIV	Drive from which DOS was booted
01375	DISKTYPE	Type of disk from which DOS was booted
01376	DISKADR	I/O bus address of booted disk controller
01377	DOSFLAG	DOS Flag byte #1
01200	DOSFL2	DOS Flag byte #2
01400	MCR$	Monitor Communication Region
01544	LFT	Logical File Table
4	TFT	temporary file table
0<4	LF0	logical file #0
1<4	LF1	logical file #1
2<4	LF2	logical file #2
3<4	LF3	logical file #3

Logical File Table Description

0	PFN	(1) PHYSICAL FILE NUMBER
1	PDN	(1) PHYSICAL DRIVE NUMBER AND PROTECTION
2	LRN	(2) NEXT LRN TO BE DEALT WITH
4	BLRN	(2) FIRST LRN WITHIN CURRENT SEGMENT
6	CSD	(2) CURRENT SEGMENT DESCRIPTOR
8	RIBCYL	(1) PDA (MSB) OF RIB
9	RIBSEC	(1) PDA (LSB) OF RIB
10	MAXLRN	(2) LARGEST LRN REFERENCED
12	LRNLIM	(2) RESERVED FIELD (INITIALLY ZERO)
14	BUFADR	(1) CURRENT CONTROLLER BUFFER ADDRESS
15	XXXXXX	(1) NOT USED

DOS Memory Mapping

```
000000   LDRAD$    System Loader
001000   DOSAD$    Resident DOS
004000   OVLAD$    DOS Overlays
005400   DSPAD$    CRT Write Routine
005572   KEYAD$    Keyboard Read Routine
006000   DEBAD$    DOS Debug Routine
07400    FLDAD$    DOS Function Loader
010000   CASAD$    Cassette Tape Drivers
013400   CMDAD$    Command Interpreter
017000   COVAD$    Command Interpreter Overlays
```

DOS Keyboard/Display Routine Control Byte Equates

```
3        EOS       end of string, no CR/LF
011      H         horizontal position follows
013      V         vertical position follows
015      EOL       end of line, CR/LF
021      ECF       erase cursor to end of frame
022      ECL       erase cursor to end of line
023      R         roll screen up one line
11       BL        number of bottom line on screen
0        TL        number of top line on screen
                   (not valid on all processors)
79       RC        number of rightmost column on screen
0        LC        number of leftmost column on screen
```

DOS FLAG byte #1 (location 01377)

```
1<7      ABTIF     1... .... abnormal program completion
1<6      NETACT    .1.. .... ARC active
1<5      UBOOT     ..1. .... DOS was booted from disk
1<4      CHACT     ...1 .... chaining active
1<3      IS55AVL   .... 1... 5500 instructions available
1<2      PSACT     .... .1.. PS active
1<1      RAMAVL    .... ..1. RAM display available
1<0      ROMBOOT   .... ...1 BOOTSTRAP loaded from ROM
```

DOS FLAG byte #2 (location 1200)

```
1<7      $FL2B7    1... .... reserved
1<6      $REMARC   .1.. .... Remote console facility active
1<5      $1800CPU  ..1. .... Running on an 1800 processor
1<4      $UPSACT   ...1 .... UPS active
1<3      $MEMD     .... 1... memory resident overlays destroyed
1<2      $MEMU     .... .1.. memory resident overlays present
```

```
1<1       $FL2B1     .... ..1. reserved

1<0       $ONESHOT   .... ...1 One-shot bit for startup procedures
DISKTYPE (location 01375)

  0       $ARC       ARC remote volume
  1       $9370      9370 25MB mass storage disk
  2       $9374      9374 20MB disk
  3       $9390      9390 storage module system
  4       $9350      9350 2.5MB disk
  5       $RESVAL    reserved
  6       $9380      9380 single-density diskettes
  7       $1840      1842 dual-density diskettes
  8+                 reserved
```

CHAPTER 58. PROCESSOR DEBUG

58.1 Introduction

The Datapoint 5500, 6000, 6600, 1800, and 3800 family processors include a DEBUG program implemented in ROM, whose immediate accessibility creates a flexible interface between user and machine. This guide is intended to provide the user of these systems with that information essential to the use of the ROM-DEBUG. With this powerful hardware feature the user should quickly develop an aggressive debugging tool.

58.2 Startup Procedure

There are five methods of entry to DEBUG:

(1) Forcing entry through manual intervention.
(2) Entry through a BREAKPOINT set by DEBUG.
(3) Entry through a BREAKPOINT imbedded in the user Program.
(4) Entry as the consequence of a RETURN from a DEBUG Call Command.
(5) Entry through a hardware vector such as "E1 MEMORY PARITY ERROR".

To force entry to DEBUG, depress in sequence the DISPLAY, RUN, and RESTART keys (DSP, INT, and RESTART on an 1800/3800), keeping each key depressed until all three are down. Then release RUN (INT on an 1800/3800). This will bring up the DEBUG display and commands may be entered.

Note that depression of the DISPLAY key during the transition from Boot Block read-in to execution during REBOOT will also cause entry into DEBUG.

58.3 Saving the Machine State

When DEBUG is entered through console intervention, most of the user's program state is undisturbed. Information not saved is the state of the interrupt enable flip-flop (interrupts are disabled), the state of the base register or sector table (these two are not changed upon entry to DEBUG), the state of the ALPHA/BETA Mode flip-flop (all registers are saved), the state of the I/O system (what device is addressed and the state of its status/data select flip-flop), and the bottom two stack locations.

Information saved is the ALPHA/BETA Mode registers and condition code flip-flops, the Program Counter (PC) and 016 Stack locations.

Note that there exist default values upon exit from DEBUG for:

(1) ALPHA/BETA Mode flip-flop
(2) Currently addressed device and its Status/Data Mode flip-flop
(3) Interrupt enable flip-flop (always disabled)

The first two of these can be set using DEBUG commands ('A', 'G' and 'R').

58.4 Display Format

The ROM-DEBUG display consists of four lines and occupies the bottom-right corner of the screen.

```
      AAAAAA     : CURADR
      *  NNN     : ASCII,8 BIT OCTAL C[CURADR]
      MMMMMM     : LSB, MSB ADDRESS FORMED AT CURADR.
nnnnnnn*         : COMMAND INTERPRETER
```

The first (top) line shows the current sixteen bit address.

The second line contains both an ASCII (One character shown as *) and an 8-bit octal (Three characters shown as NNN) representation of the contents of the current address byte.

The third line contains an octal representation of the 16-bit value whose LSB is at CURADR and whose MSB is at CURADR+1. (This is the address format used by JMP, CALL and DA mnemonics).

58.5 The Command Interpreter

The bottom line of the display is an interpreter used to edit and input commands to DEBUG. The blinking cursor signifies that the Command Interpreter is awaiting user input.

Data is entered serially into the input display buffer. The cursor is displaced to the right successively as this occurs. The Backspace key erases the character most recently entered, shifting the entry cursor to the left one space. The cancel key deletes the entire entry.

All commands are single characters. Commands which accept input arguments are preceded by the argument, which is entered in octal. Not all commands require an input argument. The last character input to the interpreter must be a legal command. Illegal input is ignored, evoking a BEEP from the processor. Commands are executed upon their entry into the interpreter (no ENTER key is required), with the current contents of the entry line being cleared. Upon command completion the cursor reappears, awaiting further input.

58.6 Command Syntax

This explanation of the command syntax uses the following notation:

nnnn... Indicates an optional sequence of octal digits not to exceed the number of n's given.

If input argument contains more than eight bits of significance, special results will occur. In general what will happen is that two bytes of memory will be affected by the command, either a register pair or a memory address in LSB, MSB format.

12345 There exists a set of special commands whose accidental execution is inhibited by the requirement that they contain this unique argument.

58.7 Input Command List

nnn A Address the given or current I/O device. No check is made on address format. STATUS is displayed as C[CURADR]. NOTE that the CURRENT Device is readdressed and put into the mode last accessed (Data mode if 'F' or 'G' have been executed subsequent to last 'A' command) prior to resumption of execution through Call, Exit, Jump or User Exit Commands if the last I/O DEBUG command executed is A.

nnnnnn B Set a BREAKPOINT at the given or current address. Upon BP execution the state of the machine is saved, the memory location at which the BP was set is restored to its original value and the corresponding BP table entry is cleared.

The following notes reference the use of the 'B' command.

Overlay BREAKPOINT will not loop. That is: It is not possible to successively set a BREAKPOINT in the same memory location in order to iterate the execution of a program loop. To iterate BREAKPOINT through looping sequence requires 'double Breakpoint'. Twenty BREAKPOINTs can be active at any one time. Note that BP's DISABLE INTERRUPTS and leave them disabled prior to resumption of execution through Call, Exit, Jump or User Exit commands. This is done to enable testing of Foreground routines with DEBUG. If it becomes necessary to use DEBUG with Interrupts Enabled, the user should place an EI instruction in a main loop of his program. It is impossible for the machine to determine its current register (ALPHA/BETA) mode. Therefore the 'R' command mode flip-flop is set to ALPHA when a BP is encountered. If the user wishes to test code written in BETA Mode it is necessary that he manually put the Machine in BETA Mode (With 'R' Command) prior to resumption of execution through Call, Exit, Jump or User Exit commands. Similarly, he may have to address the proper I/O device (with A) and perhaps put it into DATA Mode (with G) before continuing execution from a breakpoint. Note that DEBUG will not set a BREAKPOINT over another BREAKPOINT.

nnnnnn C Call the given or current address. The Machine State is restored before execution control is passed to the

Subroutine. A RETURN from the Called Subroutine causes re-entry into DEBUG and hence, causes the Machine State to again be saved.

D Decrement the current address value. Any Input Argument will be ignored.

E Continue execution from a forced or BREAKPOINT entry into DEBUG. Machine State is restored prior to resumption of execution. The interrupts are left disabled. The register mode is set to the last R value (initialized to ALPHA Mode upon BP or on forced entry), the base register and sector table are not changed, and the I/O device is addressed and optionally set to DATA mode. Note that this command does not depend on any Display Parameters. Prior DATAPOINT Debug software used CURADR as an exit address pointer.

nnn F Fetch next data byte from current or given I/O device. Command will automatically put device in DATA Mode and the device will subsequently be put in data mode when the E command is given.

nnn G Go to data mode in the current or given I/O device when the E command is given.

H * Not Used. *

I Increment the current address value. Any Input Argument will be ignored.

nnnnnn J Jump to the given or current address. Machine State is restored prior to resumption of execution.

12345K Set ASCII keyin mode. Will allow ASCII data to be entered into CURADR in auto-increment mode (i.e. will update CURADR). BACKSPACE moves CURADR back and displays its contents. DELete moves CURADR forward and displays its contents. CANCEL causes a return to normal mode.

L Link to the address pointed to by the Current Address. CURADR is replaced by line 3 (the 16-bit LSB, MSB address formed at CURADR,CURADR+1). The remaining display parameters are updated appropriately. Note that initial display state upon entry into DEBUG can be regenerated by performing the 'S' command, followed immediately by the 'L' command.

(nnn)nnn M Modify the contents of the current address location. If the value of the Input Argument exceeds eight bits of significance, two memory locations will be modified, treating the input argument as an address in LSB, MSB Format for JMP and DA. (A CLICK is sounded to notify the operator of this action.)

N * Not Used. *

O * Not Used. *

nnnnnn P Load the Base Register with the 8-bit value (nnnnnn - 0100000)

12345Q Load the Sector Table. CURADR => Table whose first byte equals the number of entries to be loaded. The following bytes contain arguments to be loaded into the Sector Table.

R Switch Alpha/Beta Mode register display. The ASCII character displayed after command execution tells the current display mode: A=ALPHA, B=BETA.

nn S Display the specified stack item (up to 015 Octal). Note: P, 0 => 014 Octal after RESTART. (Since RESTART PUSHes P onto the top of the STACK.)

12345T Start memory test. Displays Memory Size and Pass Counter in right-bottom corner of screen. Maintains running display of Test Failures.

U User mode execute. Command sets USER Flag then executes 'E' Command.

nnn V EX COM4
nnn W EX WRITE
nnn X EX COM1
nnn Y EX COM2
nnn Z EX COM3

The various EX commands affect the device specified by the last A command. The nnn entry specifies the output byte value issued with the command. Following the command, the DEBUG display shows the device status

Set Current Address to nnnnnn. Command has no efect unless it is preceeded by an Input Argument.

<Cancel> Cancel entry line.

<BSP> Backspace on entry line.

(nnn)nnn . Modify the contents and then increment the current address. If input argument has more than eight bits of significance, two memory locations are modified, treating the argument as an address in LSB, MSB Format. (a CLICK is sounded).

(nnn)nnn ^ Modify the contents and then increment the current address. If input argument is null, the last non-null value given is used. If 'last value' exceeded eight bits of significance, two memory locations will be modified. (a CLICK is sounded).

Clear all active (DEBUG set) breakpoints, restoring values.

(nnn)nnn a
nnn b
(nnn)nnn c
nnn d
(nnn)nnn e
nnn h
(nnn)nnn l
nnn x

Display or modify register or register pair. If no input argument is provided, the register contents are displayed (center line of display). Specify LSB register of a register pair (L for HL pair) to display pair contents on bottom line. If an input argument is provided the register contents are modified and then displayed. If input argument exceeds one byte, a register pair is modified.

f Display the condition flags. The byte displayed is structured:
7=>C, 6=>S, 1=>-Z&-S, 0=>-Z&-P
This value added to itself will generate the flag values it represents.

58.8 DEBUG Command Summary

```
     nnn A - Address the given or last I/O device
  nnnnnn B - Set a break point at the given or current address
  nnnnnn C - Call the given or current address
         D - Decrement the current address
         E - Continue execution
     nnn F - Fetch the next data byte from current I/O device
     nnn G - Go to data mode in the current I/O device
         I - Increment the current address
  nnnnnn J - Jump to the given or current address
         K - Set ASCII keyin mode (12345K)
         L - Link to the address pointed to by the current address
(nnn)nnn M - Modify the contents of the location ponted to by the
             current address
  nnnnnn P - Load the page basing register
         Q - Load the sector table
         R - Switch from alpha to beta mode or vice versa
      nn S - Display the specified stack item
         T - Start memory test ('12345T')
         U - User mode execute
     nnn V - EX COM4 to last I/O device
     nnn W - EX WRITE to last I/O device
     nnn X - EX COM1 to last I/O device
     nnn Y - EX COM2 to last I/O device
     nnn Z - EX COM3 to last I/O device
(nnn)nnn a - Display or update the contents of the A-register
     nnn b - Display or update the contents of the B-register
(nnn)nnn c - Display or update the contents of the C-register
     nnn d - Display or update the contents of the D-register
(nnn)nnn e - Display or update the contents of the E-register
         f - Display the flags (adding the number to itself will
             restore the flags)
     nnn h - Display or update the contents of the H-register
(nnn)nnn l - Display or update the contents of the L-register
     nnn x - Display or update the contents of the X-register
(nnn)nnn . - The equivalent of an M followed by an I
  nnnnnn <enter> - Change the current address
         # - Clear breakpoints
(nnn)nnn ^ - Modify and Increment using last value
```

ROM DEBUG DISPLAY

AAAAAA - The current address (in octal)

AAAAAAA
X NNN
MMMMMM

X - The contents of location AAAAAA (in ASCII)
- or the contents of the specified register (in ASCII)

NNN - The contents of location AAAAAA (in octal)
- or the contents of the specified register (in octal)

MMMMMM - The contents of locations AAAAAA+1 and AAAAAA respectively, concatenated into one octal number
- or the contents of a register pair concatenated into one octal number (XA, BC, DE, HL)

58.9 Extensions to Standard DEBUG

The DEBUG described above is specifically that implemented on the 5500 processor. The other processors using ROM-DEBUG have implemented a superset of the 5500 DEBUG and recognize an expanded list of commands. For descriptions of the commands unique to each processor family, consult the Datapoint product specification for the processor in use.

APPENDIX A. DOS.A AND DOS.E

DOS.A and DOS.E are two Disk Operating Systems supporting Datapoint computers operating in conjunction with up to four 9350 series cartridge disk drives.

A.1 Planning for DOS.A/DOS.E

DOS.A and DOS.E are both alike in many respects. Both use the 9350-series disk cartridge drives, and they are each almost identical to the other operationally. The primary operational difference between DOS.A and DOS.E is that DOS.E will support the Datapoint Partition Supervisor, PS, released separately. Operating under PS, DOS.E permits the concurrent execution of more than one partition.

A.1.1 DOS.A Physical Configuration

DOS.A operates in either Datapoint 2200, 5500 or 6600 family processors with at least 16K of memory and one or more 9350-series disk drives. Use of a single 9350-series drive is possible, but a multi-drive system should be available for backup and support purposes. Some consideration must be given to the question of copying files from one disk to another, and most systems incorporating the 9350-series disks will have files large enough to make it impractical to transfer them from one disk cartridge to another one cassette at a time.

An option which should be considered during the systems planning phase is the High Speed, or so-called "RAM" Display Option for 2200 processors. This option is strongly recommended, as it can substantially increase total system throughput (especially on batch-processing oriented systems) at a very small additional cost. This option is field-installable, and is standard equipment on Datapoint 5500 and 6600 family computers.

A.1.2 DOS.E Physical Configuration

DOS.E differs from DOS.A in that DOS.E requires a minimum 48K Datapoint 5500 processor and two, three, or four 9350-series disk drives attached to a 9357 disk control unit. This enhanced cartridge disk controller contains four times the amount of high speed cache memory contained in the older 9350-series controller, as well as additional hardware features to facilitate the multiprogrammed environment available under PS/DOS.E. (Older 9350-series disk controllers can be easily field-upgraded to 9357 levels.)

A.2 Disk Drives

DOS.A and DOS.E support a maximum of four 9350-series cartridge disk drive units.

A.3 Disk Media

The Datapoint 9350-series disk drives use a single platter disk cartridge, media-compatible with the IBM 2315 disk cartridge. Data is recorded in 203 concentric circles on each of the two recording surfaces. Each such circle is referred to as a track.

The disk itself is enclosed within a plastic cartridge which helps to protect it from bumps, jolts, and contaminants while it is not in place in the disk drive. This cartridge and the care taken in its handling and storage are of prime importance in helping to eliminate disk errors and parity failures that contamination can cause.

A.4 Loading and unloading Disk Cartridges

Loading and unloading cartridges from the 9350-series drives is simplicity itself. At the top of the front side of the drive is the cartridge access door. Pulling out and down on the handle opens this door. The cartridge is inserted into the cavity with the "tongue" of the cartridge on top and entering first. When the cartridge is fully inserted, the cartridge access door is closed and the rocker switch marked "LOAD/RUN" is switched to the "RUN" position. When the switch is moved to "RUN", the following things occur:

1) The cartridge access door is locked closed.

2) The indicator lamp marked "LOAD" on the front panel of the drive is extinguished;

3) The disk pack accelerates to its rated speed of 1500 rpm, at which time the indicator lamp marked "READY" lights up. When this lamp lights up, it indicates that the disk has come on-line for the Datapoint processor.

Removing a cartridge which is no longer needed from a drive is a simple reversal of the above steps. First, the "LOAD/RUN" switch is moved to the "LOAD" position. The drive immediately goes off-line to the processor and is swiftly braked to a smooth stop. When the disk comes to a full and complete stop, the door is unlocked and the "LOAD" indicator lamp comes on. At this time, the cartridge access door can be opened with a gentle tug, after which the cartridge simply slips right out. The cartridge should be stored in a suitable storage rack; it should never be left in a place where it might slip and fall onto a hard surface, such as a floor.

A.5 Switches and Indicators

The current cartridge disk drive, manufactured by Wangco, uses a small cluster of controls in the lower right-hand corner of the disk drive front panel. There is a thumbwheel switch for physical drive number selection, which is set at installation and should not be moved thereafter. The rocker switch marked "RUN" and "LOAD" controls disk loading as described above; the "READY", "LOAD" indicator lamp is immediately below this switch. The leftmost controls are a pair of rocker switches marked "PROT CART" and "PROT FIXED". These switches control the write protection status of the cartridge disk and the fixed disk inside the drive. When the indicator lamp behind one of these switches is lit, the corresponding disk is write-protected. The protection can be changed at any time by changing the switch position.

The older cartridge drive, manufactured by Diablo, has only one single rocker switch, (the LOAD/RUN switch which has been previously described) and four color-coded indicator lamps. The first of these, a white lamp marked "LOAD" comes on to indicate that the drive is ready to have a disk cartridge inserted or removed. The second lamp, a yellow one marked "READY" indicates that the cartridge in place has come up to speed and is on-line. The third lamp, an orange one marked "CHECK", is an error indication. This lamp is rarely seen illuminated. If it does light up, taking the drive offline and back online may help (switching the LOAD/RUN switch to LOAD and back). If that does

not work, try powering down the entire system and then turning it back on again, using the main power switches. If the CHECK condition still is not cleared, call the Datapoint Customer Support Center for technical assistance. The fourth red lamp is marked "PROTECT", and when it is illuminated the processor cannot write on the disk in that drive. The disk is protected each time it is brought to RUN status. Depressing the PROTECT button extinguishes the indicator lamp and write-enables the disk. The disk can be re-protected only by switching the LOAD/RUN switch to LOAD and back to RUN.

A.6 Care and Handling of Disk Cartridges

Disk cartridges for the 9350-series disk drives are precision assemblies and must be treated with some care. It is highly important that they not be dropped, mishandled, or contaminated with dust or other pollutants. The cartridges should be stored in an appropriate storage rack, in an area free from dust and in an environment similar to that where the drives are installed (preferably in the same room with the computer). Users should be very careful to never allow anything to contact the oxide surface of the disk itself.

If the cartridges are shipped by common carrier, they should be repackaged in their original, protective shipping carton and marked "FRAGILE". Disk cartridges should never be mailed by Parcel Post. Upon receipt of a disk cartridge, if there is any evidence of damage the cartridge should not be used until it has been inspected and approved for use by a Datapoint service representative.

In addition, any cartridge which has been in a non-computer room environment should be allowed to equalize temperatures in the room with the computer for 24 hours before use if at all possible before attempting to read or write data on the cartridge. In an emergency, placing the cartridge onto a drive and letting it spin up and run for about an hour will usually be adequate, but this procedure should be considered an emergency measure only.

A little care in handling disk cartridges will repay itself several times over in reliable and trouble free service with long life from your disk cartridges.

A.7 Care and Maintenance of the 9350 Drives

As with the disk cartridges themselves, cleanliness of the 9350 disk drives is of great importance. All efforts should be made to keep the room as dust-free as possible. Since the read/write heads fly very close to the disk surface (about 100 millionths of an inch away from the oxide surface) even such small particles in the air as those present in cigarette smoke are apt to cause troubles sooner or later. Any dust that may collect around the disk drives should be regularly cleaned away.

In addition to this user maintenance, the user should also ensure that his local Datapoint service representative performs the preventive maintenance procedures outlined in the 9350 series disk drive maintenance manual. These preventive maintenance procedures can be compared to changing the oil and oil filter in the family automobile. An automobile will perform all right for a while without regular oil and filter changes, but sooner or later it will extract a heavy penalty for not having better care. The same characteristic holds true for disk drives as well.

A.8 Head Crashes

Each of the two heads in the 9350-series disk drive is held near the disk oxide surface by a spring which pushes the head toward the surface with a force of approximately 350 grams. The disk, on the other hand, is spinning at approximately 50 miles per hour relative to the head. The head and disk are kept apart by a micro-cushion of air only about 100 millionths of an inch thick. A head crash occurs when this lubricating air film fails. The main causes of head crashes are foreign particles in the lubricating film, contamination buildup on the surfaces of the disk or read/write heads, or a defective disk surface.

When a head crash occurs, the head rubs directly against the oxide surface of the disk, which frequently loosens more oxide, resulting in further and more severe crashes, and things go progressively downhill from there. Due to the severity of a head crash, not just because of the loss of data on a disk but also due to the degree of damage to the heads on the drive, it is important to recognize the symptoms of a head crash. In this manner a disk experiencing a head crash can usually be discovered and stopped before the crash reaches catastrophic proportions.

A.8.1 Prevention of Head Crashes

There are three main things that a user can do to help minimize the likelihood of a head crash. These include:

1) Preventive maintenance. Establish a preventive maintenance schedule with your Datapoint customer engineer and stick to it. Make sure that this preventive maintenance gets done. Particularly important is attention to the head/arm assemblies, air filtration system and moving parts.

2) Proper handling and storage of disk cartridges. Disks should be carefully stored in an area free from dust, smoke, and other contamination. Any disks whose cartridges are cracked or broken should be replaced immediately. Disk cartridges should be handled carefully to avoid bumping or dropping. Never insert a dropped cartridge into a drive! Give it to a Datapoint service representative for inspection.

3) Keep the cartridge access door closed. Never leave it open. The longer it is open, the greater the susceptibility to contamination.

A.8.2 Recognition of a Head Crash

In spite of all precautions, chances are that most users will experience a head crash sooner or later. Being able to identify it quickly when it happens can help to minimize the damage. A head crash may be indicated by one or more of the following symptoms:

1) Repetitive hard read or write parity errors. Because of the propagation effect of a head crash, do not move any disk with massive hard parity errors to another drive. If errors persist, then the possibility of a head crash exists and must be investigated.

2) Audible tinkling sound. An audible tinkling sound from the disk, which may progress to a screech, probably indicates a head crash.

3) Visible damage to the disk surface. Any scratch on the recording portion of the disk surface where the aluminum substrate is exposed. Concentric adjacent scratches of any length. A single scratch of over approximately three

inches in length. Imbedded particles or an accumulation of loose oxide on the surface. Any of these can indicate that a head crash has occurred.

A.8.3 What to Do if You Have a Head Crash

If you suspect that you have had a head crash, call the Datapoint customer support center at once. In the meantime, observe the following precautions:

1) The disk which was mounted on the drive when the crash occurred should be considered suspect and should not be mounted on any other drive until it has been inspected by the Customer Engineer and approved for use.

2) The drive which experienced the crash should not be used until it has been thoroughly checked by the Customer Engineer. Other disks which are probably intact can be damaged by a drive which has had a crash, since the same drive is apt to crash again with any subsequent disk placed in it until it has been properly serviced.

3) Head crashes should be considered to be contagious. A disk which has crashed may have loose oxide or other irregularities on its surface. If the disk is placed into a different drive, these contaminants are apt to very quickly result in a crash occurring on the new drive as well. Since the loose oxide or whatever can build up on the heads of the drive as well as the disk itself, the drive can carry the contaminants of a bad disk over to any number of good disks subsequently used on it, and these can in turn contaminate other drives.

A.9 Preparing Disk Packs for Use

When a disk cartridge is first received from the manufacturer, it is completely demagnetized. However, unlike the other Datapoint Corporation disk drives, on the 9350 series drives the position of the sectors on the disk surface are determined by the sector timing slots around the edge of the disk's hub. Therefore, no special preparation of the disk (other than the DOSGEN process itself which is always required) is necessary before a new cartridge can be used by the DOS.

A.10 Disk Organization under DOS.A/DOS.E

This section describes the logical organization of the data on the disk when operating under DOS.A/DOS.E and how it relates to the general DOS file concepts as described in the chapter on System Structure. In this chapter it is assumed that the user is familiar with these concepts and has read and is familiar with the basic DOS file structuring.

A.10.1 Logical Drive Mapping

Under DOS.A, each physical disk cartridge corresponds with precisely one logical drive. Since the 9350-series disk controller is only capable of attaching four 9350-series disk drives, that means that only four logical drives (numbered 0, 1, 2, and 3) are legal under DOS.A.

A.10.2 Size of a Logical Drive

Each logical drive is two tracks on each of 203 cylinders of the physical disk cartridge. This results in 406 tracks of 24 sectors each, or a total of 9,744 total sectors on a disk cartridge. Since cylinder zero is reserved for system tables, only 9,696 sectors fall into allocatable file space and therefore only 9,696 sectors are available for storage under the DOS.A file management scheme. Of these, almost 100 sectors are required for the minimum DOS.A system, the eight DOS.A system files (SYSTEM0/SYS through SYSTEM7/SYS). This leaves on the order of 9500 sectors for user data once the DOS.A proper and a few of the basic commands have been loaded.

A.10.3 Cluster Mapping

Because there are eight bits per byte in the cluster allocation table (or CAT for short), and it is desirable to maintain one byte in the CAT per cylinder of available space on the drive, each cylinder on a logical drive (containing 48 sectors, total) is broken into eight groups, each one containing six physically contiguous sectors. Each such group is called a cluster. The first four clusters per cylinder are recorded on track zero of the cylinder, and the second four clusters of that cylinder are recorded on the other side of the disk, which is track one, of the same cylinder.

Due to the fact that space is always allocated in terms of an

integral number of clusters, this implies that the minimum file size under DOS.A is six sectors and that file size will always be a multiple of this number.

A.10.4 Segments under DOS.A

Disk space under Datapoint Corporation's DOS is always allocated in contiguous chunks of clusters called segments. When space is allocated, the largest segment on the disk (up to the maximum possible sized segment) is allocated, to keep the file as free of fragmentation as possible. By limiting the allocation size to the size of a full segment, the problem of allocating all available space on a disk to a first scratch file before a second one is subsequently opened is minimized. If several scratch files are opened and space in them is allocated at regular intervals, the resultant segments will be interleaved, resulting in minimized access time as the heads randomly access throughout the scratch area. The desire to make segment size small (to minimize file space conflicts and help to optimize use of space on the disk) and yet large (to maximize processing speed, maximize file size and minimize the number of RIB accesses) resulted in a segment size of thirty-two clusters. This compromise results in a 192-sector segment (thirty-two clusters of six contiguous sectors each) allowing easy addressability of a maximum size file while still allowing the segment size information to be kept within five bits as required for RIB compatibility with the other versions of DOS.

A.10.5 Maximum File Size

Under DOS.A, the maximum file size available is about 9,600 sectors. This is because there are 9,696 allocatable sectors of which almost 100 are used for the DOS.A system files. In practice, the user should not ever construct a system which pushes against the limits of available file size on a disk, since this fails to allow for future growth and expansion of his system. Another consideration is that if any tracks need to be locked out on the disk cartridge due to surface defects, then there may not be enough space left on the disk for his file.

Files bigger than about 9,000 sectors should be kept on larger disk systems, such as 9370 series disks under DOS.B or other appropriate DOS. If files larger than that size must be kept under DOS.A, then the files should be segmented into two or more distinct files and logically concatenated at the user program level, the same as would be necessary for files larger than about 800 sectors on the 9380 series diskettes.

A.10.6 Cluster Allocation Table and Directory

Each disk cartridge used under DOS.A has its own, completely self-contained directory and file structure, just as for all Datapoint Corporation DOS. There are sixteen directory sectors on each disk cartridge, located in consecutive sectors starting at sector six on track zero of cylinder zero. Therefore, the sectors go from sector six to sector 025 (octal). The cluster allocation table is at sector zero of track zero, cylinder zero. The lockout cluster allocation table is at sector one of track zero, cylinder zero. The hashed directory index is at sector two of track zero, cylinder zero. The backup copies of each of these are in the corresponding locations of track one of cylinder zero.

The Hashed Directory Index, maintained by the DOS, resides in sector two of track zero, cylinder zero. This table enables directory lookups to go about four times faster than was possible under DOS 1.2. The technique works as follows:

Given an eleven byte file name and extension, an arithmetic/logical operation upon the file name results in an eight-bit quantity referred to as a hash code. This code is essentially a condensation of the 11 bytes of file name and extension information into only one byte. Obviously, the information is not complete; there are only 256 distinct eight-bit hash codes possible, while there are literally billions of legal file names and extensions. However, the condensation of information is such that looking at the hashed directory index allows the DOS to substantially restrict the range of directory sectors it must examine when doing a directory lookup. Each hash code for the file names in the directory is stored into the hashed directory index, offset by the physical file number (PFN) of the file with the corresponding name and extension.

Note that there is a calculated danger in the hashed directory approach. The danger is that if the hashed directory index is overwritten or otherwise destroyed accidentally, files may become inaccessable even though they are clearly shown (by doing a CAT command on the disk, for example) to be present. When this occurs on a disk, the technique to repair the disk is the REPAIR command. When the REPAIR command is almost finished, specify that the Hashed Directory Index is to be rewritten to the disk. This causes the HDI to be regenerated from the actual disk directory and rewritten. In general, the Hashed Directory Index is rarely if ever destroyed in actual disk usage, and contributes greatly to overall system performance if many directory lookups are being done.

A.11 Internal DOS Parameterization

This section describes the DOS.A-dependent details of the parameterization of DOS.A system routines.

A.11.1 Physical Disk Address Format

Under DOS.A, physical disk addresses are presented (for example, as input to the DR$ and DW$ routines) in a two-byte format quite similar to that used under the other DOS. The most significant byte (which is traditionally placed in the D register) is the cylinder number, just like for DOS.B and DOS.C. The less significant byte (usually placed in the E register) has its most significant three bits representing a cluster number within the cylinder (any combination of these three bits is valid) and the least significant five bits representing a relative sector number within the specified cluster. Only the values zero through five are valid for the least significant five bits, since there are only six sectors per cluster.

A.11.2 Hardware Address Structure

The hardware disk address for 9350 disks also requires two bytes. One byte specifies cylinder number. The other byte specifies a sector number, 0 - 027 on the bottom surface, 040-067 on the top surface. This hardware address is used only for the DUMP9350 program and internally to the DOS routines DR$ and DW$.

APPENDIX B. DOS.B

DOS.B is Datapoint Corporation's Disk Operating System supporting Datapoint 2200, 5500 and 6600 family computers operating in conjunction with up to two 9370 series disk drives.

B.1 Planning for DOS.B

The recommended configuration for a DOS.B system includes 16K or more of memory in the 2200 or 5500 series computers. Use of a single 9370-series drive is possible, but the user should at least have access to a double-drive system for backup purposes. Some consideration must be given to the question of copying files from one disk pack to another, and users of the 9370-series "Mass Storage" disk systems will typically have files far too big to consider transferring from one disk to another one cassetteful at a time.

Another option which should be strongly considered is the High Speed, or so-called "RAM" Display Option for 2200 processors. This option can substantially increase total system throughput (especially on batch-processing oriented systems) at a very small additional cost. The RAM Display option is field-installable, and is standard equipment on Datapoint 5500 and 6600 family computers.

B.2 File Storage Capacity under DOS.B

Under DOS.B, each 9370-series disk unit is dealt with as two logical drives. Each of these two logical drives contains 38,976 sectors of 256 bytes each and can store up to 256 files. Of these, about 250 sectors and about ten files are used by the operating system and a few basic commands, leaving about 10 million bytes of usable space per logical drive, or up to roughly 20 million bytes of storage total for each disk storage unit in the configuration.

Other features of DOS.B include a large maximum file size: up to 30,237 data sectors in a single DOS.B file (not including the end-of-file mark and two RIBs).

B.3 Disk Drives

Datapoint DOS.B supports one or two 9370-series disk drives attached to one 9370-series disk controller.

B.4 Disk Media

The Datapoint 9370 series comprises two different types of drives. Models 9370-9373 use an 11-platter disk pack, media-compatible with the IBM 2316 disk storage module. On these packs data is recorded in 203 concentric circles on each of the 20 recording surfaces. Each such circle is referred to as a track. Models 9374 and 9375 use a single-platter disk which records data on 408 tracks (DOS uses only 406 of these).

The disk pack is enclosed within a plastic enclosure when it is not in place in the drive. This cover is intended to help keep the disk free from dust, pollen, smoke and other contaminants and is of prime importance in helping to eliminate disk errors and parity failures that contamination can cause.

B.5 Loading and unloading Disk Packs

B.5.1 Models 9370-9373

On the right side of the top of the 9370-series disk drives is the disk access cover. While holding the disk pack by the top center handle, remove the bottom portion of the disk pack enclosure by turning the bottom knob with the other hand. Then raise the disk access cover and carefully lower the disk pack into the cavity, still holding the disk pack by the top handle. When the pack has fully seated onto the spindle, turn the disk pack top center handle fully clockwise, until firm resistance is met. It is important that the pack be solidly in place before removing the top cover. After the pack has been properly mounted, the top cover should be slowly and carefully removed by lifting it straight upwards. Avoid letting the cover tilt and wedge against the edges of the disk platters as it is being drawn upwards as this can affect the precision alignment of the disk pack. The access cover should be closed as soon as the disk pack cover has been fully removed, and the top and bottom halves of the disk pack protective cover should be immediately put back together to keep out dust and other contaminants.

To remove a disk pack, first place the "START/STOP" switch on the operator control panel of the drive to the STOP position. This immediately takes the drive off line and activates dynamic braking circuits in the drive which will brake the pack to a smooth but rapid stop in about twelve seconds. The disk access cover on top of the drive must not be opened before the pack has come to a full and complete stop. When this has occurred, raise the disk access cover and carefully lower the top portion of the disk pack cover down onto the pack. Be certain not to get it skewed since if the cover wedges against the edges of the platters it is possible to affect the critical surface-to-surface alignment of the pack, which will damage it. When the cover is fully lowered onto the pack, turn the handle in the center of the top of the cover counterclockwise until a distinct click is heard. This click indicates that the pack has been released from the drive spindle and may now be removed. Lift the disk pack and top cover together carefully out of the drive and immediately reattatch the bottom cover to the base of the pack, locking it firmly in place by a twist of the knob in the center of the bottom portion of the canister. The pack should be stored horizontally on a shelf (never on edge!) and in a position where it is not apt to be dropped or pushed accidentally over an edge. If another disk is not to be mounted immediately into the drive the pack was just removed from, the disk access cover should be closed right away to help prevent the entrance of dust, smoke or other contaminants into the drive mechanism and access arm assembly.

B.5.2 Model 9374/9375

At the top of the front panel of the drive is a handle for access. Pull forward and down on this handle to release the drive, then slide the entire drive forward to expose the cavity in which the disk fits. The disk pack itself has a handle on the top of the case. To open the disk pack, place the handle folded flat against the case and slide the lock button to the left, then - holding the lock button on - lift the handle to its full vertical position. This action releases magnetic clamps and allows the bottom of the disk cover to fall off. Lower the disk into the cavity in the drive, being sure it is fully seated. Now lower the handle on the top of the disk container. Invert the bottom of the disk pack cover and place it on top of the disk, inside the drive. It is essential the disk pack bottom cover be placed in the drive, since the disk will not run if the cover is not present. Finally, slide the drive back into its cabinet, closing the access door.

Removing a disk is the exact reverse of inserting it. To remove the disk from the cavity in the drive, the lock button on the handle must be held to the left, just as for opening the disk

cover.

B.6 Switches and indicators

B.6.1 Models 9370-9373

Two types of drives are repesented in these model codes; both use the same controller and the same disk packs. Some are "Telex" drives, manufactured by TSS; others are Memorex drives. The Memorex drives have no latch on the disk access cover, while the Telex drives use a spring-loaded cover held down by a latch. Both types of drive units use the same 9370 controller and provide identical performance.

B.6.1.1 Memorex Drives

The large physical drive number (just to the right of the READ-WRITE/READ ONLY switch) lights up when the heads are loaded onto the disk surface and typically at this time the drive will be on line.

The smaller numbers to the right serve as an indication of the position of the heads as they perform cylinder seeks to positions nearer or farther from the center of the disk pack. The exact physical cylinder number to which the heads are positioned at any given time can be determined by adding together the numbers which are illuminated, giving a cylinder number in decimal; the cylinder number in octal can be determined by noting which of the eight number positions are illuminated and considering those illuminated to be "1" bits and those not illuminated to be "0" bits. The bits then can be converted easily to a three-digit octal number by grouping them in groups of 2,3,3: a technique familiar to users conversant with octal.

The words "READ ONLY" illuminate to indicate that the drive is in the so-called "Write Protected" mode. In this mode, the computer cannot write anything onto the disk in that drive, but can only read the information already on the pack. This light is the indication of whether a drive is write-protected or not, and does not always immediately reflect the position of the read-only switch. See "Common Features" below.

The words "FILE UNSAFE" light up when the safety circuits in the drive detect one or more of about a dozen different conditions

that they consider would endanger the data on the disk pack if continued disk operation were attempted. The FILE UNSAFE condition can be caused by (among other things) unusually severe power surges, and infrequently by a program going completely haywire and giving flagrantly illegal commands to the disk drives. If this light comes on during use of the system, the first remedy to try is to push the switch marked "START/STOP" to the "STOP" position. After the disk has come to a complete, braked stop (which should take about twelve seconds), push the switch back to the "START" position. If the problem which caused the FILE UNSAFE condition to occur was spurious, the drive will power back up normally and come on-line again in about sixty seconds. If the FILE UNSAFE condition occurs again (usually immediately upon completion of the sixty-second power-up delay) and repeatedly, it probably indicates a hardware malfunction and time to call the Datapoint Customer Support Center.

B.6.1.2 "Telex" Drives

The controls and indicators on Telex drives are essentially identical to those on the Memorex drives. When the drive is on-line, a green indicator light comes on indicating "FILE READY". There are no indicator lights for head position; cylinder position of the heads can be read on a vernier scale mounted on the top of the access arm assembly and visible through the top of the loading cover. A white indicator lamp indicates "READ ONLY" when the drive is protected, and, as on the Memorex drives, the read/write status of the drive does not immediately reflect the setting of the READ/WRITE - READ ONLY switch (see "Common Features" below). A red indicator lamp indicating "SELECT LOCK" is equivalent to the "FILE UNSAFE" indicator on the older drives.

B.6.1.3 Common Features

Changing the setting of the READ WRITE/READ ONLY switch only affects the drive if the drive is deselected. Therefore, this switch should be normally kept in the READ-WRITE position except for special purposes, and should usually be returned to the READ-WRITE position as soon as possible after the special purpose is completed. If the DOS command interpreter is active (as indicated by the familiar DOS "READY" message) and the READ ONLY lamp and the READ-WRITE/READ ONLY switch do not concur (indicating that that drive is selected), simply entering a blank from the system console is a simple technique to ensure that the drive becomes deselected so that the revised setting of the READ ONLY switch will take effect.

The other switch on the operator control panel on the 9370-series drive (marked ENABLE/DISABLE) is for use only by the customer engineer and should always be left in the position marked ENABLE. This switch, when set to DISABLE, takes the drive off line (although the drive ready indicator stays illuminated). The switch is only active when the drive is de-selected, and this is the reason why the switch should not be used casually. If drive zero, for example, is DISABLEd and then the computer is bootstrapped, drive zero will not be de-selected at least until the DOS has completely booted itself up. And until the drive is de-selected, turning the switch to the ENABLE position has no effect. The only solution for this problem if it occurs is to completely power down the entire system and bring it back up again with the switch in the ENABLE position. So in general, it is a good idea to keep this switch set to ENABLE and let it go at that.

B.6.2 Model 9374/9375

The 9374 disk drives are controlled by a small cluster of switches in the lower right-hand corner of the front panel of each drive. A thumbwheel switch sets the physical device number; this switch is set at installation and should not be reset thereafter.

A LOAD/RUN rocker switch controls loading the disk. An indicator light below the rocker switch indicates if the drive is in LOAD, ready for a disk to be removed or inserted, or in RUN, on-line to the processor.

A pair of rocker switches labeled PROT CART and PROT FIXED control write protection of the removable disk pack and the fixed platter within the drive. When one of these switches is illuminated the corresponding disk is write protected. The protection setting of either disk can be changed at any time by using the PROT switch. When a disk is first spun up (first brought to READY status) both disks will be write protected for at least three minutes to ensure thermal stabilization. If the drive is cold, the write-protect delay could be longer. The delay for thermal stabilization is necessary because the 9374 disk is a very high-density storage medium.

B.7 Care and Handling of Disk Packs

Disk packs for the 9370 series disk drives are precision, high-technology assemblies and must be treated as such. It is of extreme importance that they not be mishandled, dropped, or contaminated with dust or other pollutants. The packs should be stored strictly horizontally (not on edge) on a shelf clean from dust and in an environment similar to that where the drives are installed (preferably in the same room with the computer).

On the bottom of each 11-platter disk pack is a fine nylon filter, normally white. This filter should be replaced at least once per year, or more often if indicated by discloration or airborne debris.

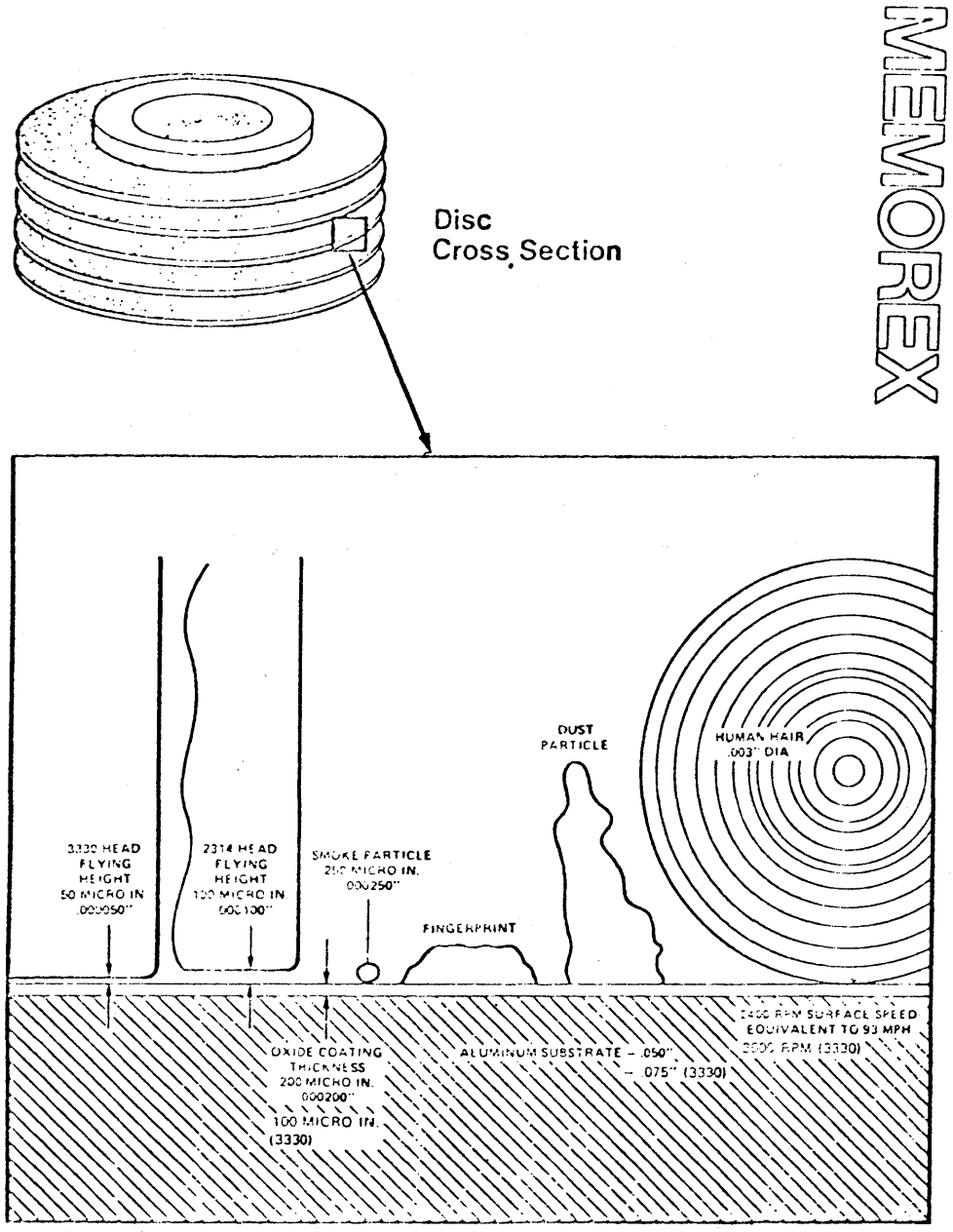
If the packs are shipped by common carrier, they should be repackaged in their original shipping carton and marked "FRAGILE". Disk packs should never by mailed by Parcel Post.

In addition, any pack which has been in a non-computer room environment should be allowed to equalize temperatures in the room with the computer for 24 hours before use if at all possible before attempting to read or write data on the pack. In an emergency, placing the pack onto a drive and letting it spin up and run for about an hour will usually be adequate, but this procedure should be considered an emergency measure only.

A little care in handling disk packs will repay itself several times over in reliable and trouble free service with long life from your disk packs.

B.8 Care and Maintenance of the 9370 Drives

The 9370 series disk drives are full scale mainframe computer peripherals and deserve to be taken care of. As with the disk packs, cleanliness is of paramount importance. All efforts should be made to keep the room as dust-free as possible. Since the read/write heads fly very close to the disk surface (about 80 millionths of an inch away from the oxide on the 11-platter packs) even such small particles in the air as those present in cigarette smoke may cause troubles eventually. The drawing in this section, reproduced here courtesy of Memorex Corporation, graphically depicts the relative proportions of disk head flying height and common office pollutants, and should help to explain why the need for cleanliness and good housekeeping practices is so important.



Users of the system should be careful to close the disk access cover (or slide the drive back in the cabinet) as soon as the pack loading or unloading is complete and keep disk packs in their protective covers at all times to prevent contamination. If disk pack canisters become soiled, they should be cleaned carefully with a mild detergent solution and carefully wiped dry. Users should be very careful to never allow anything to contact the oxide surface of the disk pack itself.

In addition to this user maintenance, the user should also ensure that the preventive maintenance procedures outlined in the

9370 series disk drive maintenance manual are performed.

B.9 Head Crashes

Each of the heads in the 9370-series drive is held near the disk surface by a spring which pushes the head toward the surface with a force of about 350 grams. The disk on the other hand is spinning at about 80 miles per hour relative to the heads. The head and disk are kept apart by a micro-cushion of air only eighty millionths of an inch thick. A head crash occurs when this lubricating air film fails. The main causes of head crashes are foreign particles in the lubricating film, contamination buildup on the surfaces of the disk or read/write heads, or a defective disk surface.

When a head crash occurs, the head rubs against the oxide surface of the disk, which frequently loosens more oxide, resulting in further and more severe crashes, and things go progressively downhill from there. Due to the severity of a head crash, not just because of the loss of data on a disk but also due to the degree of damage to the heads on the drive, it is important to recognize the symptoms of a head crash. In this manner a disk experiencing a head crash can usually be discovered and stopped before the crash reaches catastrophic proportions.

For a description of symptoms of a head crash and appropriate prventive and restorative action, see Appendix A under "Head Crashes".

B.10 Preparing Disk Packs for Use

When a disk pack is first received from the manufacturer, it is completely demagnetized and is not usable until it has been formatted. The formatting process writes the entire surface of the disk pack with track and sector addresses which later allow the controller to identify where a given sector is on the surface of the disk.

When this information is later read by the DOS, any errors discovered in the formatting information are treated in the same way as a parity error in the written sector information itself, thus resulting in up to nine or ten re-tries before returning with a parity error indication. Sometimes, if the parity error indicated by the DOS is due to an error developing in the formatting information on the disk, the parity error on the disk can be completely eliminated by using the BACKUP command to save

as much of the information on the pack as possible and then reformatting the pack. (After reformatting the pack, any data that had been on the pack is destroyed and it must be DOSGENed like a new one). Even what at first appear to be "hard" parity errors can occasionally be cleared this way.

Formatting information is written onto the disk using the INITDISK command, either from a working DOS or from a LOAD & GO cassette. (NOTE: although in general the commands from the DOS cannot be run without the DOS being active, INITDISK is one of the few exceptions). Following successful completion of the INITDISK command program, the disk pack it was used on can be DOSGENed (twice, once for each of the two logical drives) and will normally not need to be re-formatted again for the duration of its lifetime.

B.11 Disk Organization under DOS.B

This chapter describes the logical organization of the data on the disk when operating under DOS.B and how it relates to the general DOS file concepts as described in the chapter on System Structure. In this section it is assumed that the user is familiar with these concepts and has read and is familiar with the basic DOS file structuring.

B.11.1 Logical Drive Mapping

Under DOS.B each physical volume is broken into two logical drives. This is done for reasons of addressing. It is simply not possible to address all of the sectors on an entire 9370 disk volume using only two bytes of physical disk address, and the two byte physical disk address is central to all of Datapoint Corporation DOS's operations. It is not practical to change this characteristic without making changes which would result in invalidating many user-written programs and many large systems which run under the DOS. Therefore the disk was broken into two halves, and one bit of the effective physical disk address is taken from the logical drive number.

For the 9374 drives the removable disk is one logical drive, and the fixed disk is a second logical drive.

The first eight recording surfaces on the 11-platter disk (heads are numbered from zero to nineteen starting at the top of the disk drive) correspond to logical heads zero to seven on the even logical drive, and the next eight recording surfaces on the

disk correspond to logical heads zero to seven on the odd logical drive (physical heads eight through fifteen). Physical heads sixteen through nineteen (and the corresponding recording surfaces on the disk pack) are not used by DOS.B.

B.11.2 Size of a Logical Drive

Each logical drive is eight tracks on each of 203 cylinders of the physical drive. This results in 1624 tracks of 24 sectors each, or a total of 38,976 total sectors on a logical drive. Of these, about 38,400 remain when the DOS has been generated, the system tables constructed on the disk, and a few basic commands loaded. The 9374 disks are addressed using the same structure of 203 cylinders, 24 sectors per track. Physically the 9374 disks have 406 cylinders with 48 sectors per track, but the disk controller itself provides an interface between the physical and logical structure, so the processor "sees" a drive structured just like a 9370 drive.

B.11.3 Cluster Mapping

Because there are eight bits per byte in the cluster allocation table (or CAT for short), and it is desirable to maintain one byte in the CAT per cylinder of available space on the drive, each track on a logical drive represents one DOS cluster, and is represented in the CAT by exactly one bit. Since the DOS uses eight tracks per logical cylinder, this results in exactly eight clusters per cylinder of twenty four sectors each.

Due to the fact that space is always allocated in terms of an integral number of clusters, this implies that the minimum file size under DOS.B is twenty-four sectors and that file size will always be a multiple of this number. It turns out that choosing a full track as the smallest allocatable unit of space has other advantages as well from a system standpoint, since it allows some programs (like DOS.B COPY) to make several simplifying assumptions about the data in a file which enables them to copy data and reference information in a file substantially more easily and efficiently than would be otherwise possible.

B.11.4 Segments under DOS.B

Space under Datapoint Corporation's DOS is always allocated in contiguous chunks of clusters called segments. When space is allocated, the largest segment on a drive up to the maximum possible sized segment is allocated, to keep the file as free of fragmentation as possible. By limiting the allocation size to the size of a full segment, the problem of allocating all available space on a disk to a first scratch file before a second one is subsequently opened is minimized. If several scratch files are opened and space in them is allocated at regular intervals, the resultant segments will be interleaved, resulting in minimized access time as the heads randomly access throughout the scratch area. The desire to make segment size small (to minimize file space conflicts and help to optimize use of space on the drive) and yet large (to maximize processing speed, maximize file size and minimize the number of RIB accesses) resulted in a segment size of ten clusters. This compromise results in a 240-sector segment (ten clusters or tracks of 24 sectors each) allowing a maximum file size of over 30,000 sectors while still allowing internal disk address and segment size calculations to be done using faster single precision arithmetic techniques.

B.11.5 Maximum File Size

Under DOS.B, the maximum size file available is 30,238 data sectors. This number is the result of 126 segment descriptors in the RIB, each of which points at one segment of 10 tracks of 24 sectors each:

24 sectors x 10 tracks x 126 segments = 30,240 total sectors

Since the first two sectors of each file under the DOS are used for the RIB and its copy, that leaves 30,238 sectors available to the user for the storage of his data. Files longer than this number will have to be segmented or logically concatenated at the user program level, the same as would be necessary for files larger than about 9600 sectors on the 9350 series disks.

B.11.6 Cluster Allocation Table and Directory

Each logical drive under DOS.B contains its own directory and cluster allocation table, just as for all Datapoint Corporation DOS. There are sixteen directory sectors on each logical drive, located in consecutive sectors starting at sector five on logical track zero of cylinder zero. Therefore, the sectors go from sector five to sector 024 (octal). The cluster allocation table is at sector zero of logical track zero, cylinder zero. The lockout cluster allocation table is at sector one of logical track zero, cylinder zero. The backup sectors for all of these are in the corresponding locations on logical track one of the same cylinder.

The Hashed Directory Index, maintained by the DOS, resides in sector two of track zero, cylinder zero. This table enables directory lookups to go about four times faster (on full disk directories) than was possible under DOS.B Version 1. The technique works as follows:

Given an eleven byte file name and extension, an arithmetic/logical operation upon the file name results in an eight-bit quantity referred to as a hash code. This code is essentially a condensation of the 11 bytes of file name and extension information into only one byte. Obviously, the information is not complete; there are only 256 distinct eight-bit hash codes possible, while there are literally billions of legal file names and extensions. However, the condensation of information is such that looking at the hashed directory index allows the DOS to substantially restrict the range of directory sectors it must examine when doing a directory lookup. Each hash code for the file names in the directory is stored into the hashed directory index, offset by the physical file number (PFN) of the file with the corresponding name and extension.

Note that there is a calculated danger in the hashed directory approach. The danger is that if the hashed directory index is overwritten or otherwise destroyed accidentially, files may become inaccessable even though they are clearly shown (by doing a CAT command on the disk, for example) to be present. When this occurs on a disk the technique to repair the disk is the REPAIR command. When the REPAIR command is almost finished, simply specify that the Hashed Directory Index is to be rewritten to the disk. This causes the HDI to be regenerated from the actual disk directory and rewritten. In general, the Hashed Directory Index is rarely if ever destroyed in actual disk usage, and contributes greatly to overall system performance if many directory lookups are being done.

B.12 Internal DOS Parameterization

This section describes the DOS-dependent details of the parameterization of DOS.B system routines.

B.12.1 Physical Disk Address Format

Under DOS.B, physical disk addresses are represented in a two-byte format in a manner quite similar to that used under the other DOS. The most significant byte (which is traditionally placed in the D register) is the cylinder number. The less significant byte (usually placed in the E register) has its most significant three bits representing a cluster number within the cylinder (or logical track number in the specific case of DOS.B) and the least significant five bits representing the sector number within the specified cluster. Since there are 24 sectors in a cluster, the five bit sector number has a valid range of 0-027.

B.12.2 Hardware Address Structure

The 9370 series drives use three bytes for hardware addressing. The cylinder number is one byte (range 0-0312), the head number is a second byte (range 0-023), and the sector number a third byte (range 0-027).

APPENDIX C. INTRODUCTION TO DOS.C

DOS.C is Datapoint Corporation's Disk Operating System supporting Datapoint 1100, 2200, 5500 and 6600 series processors operating in conjunction with up to four 9380 series diskette drives.

C.1 Planning for DOS.C

The minimum configuration for a DOS.C system includes 16K of memory in the 1100 or 2200 series computers and 16K or more in 5500 or 6600 series computers. Use of a single 9380-series drive is possible, but the user should at least have access to a double-drive system for backup purposes. Some consideration must be given to the question of copying files from one diskette to another. Users with 2200, 5500 or 6600 series computers (having cassette tape drives) can use cassettes if necessary as a standard exchange medium for file transfers (except for files bigger than about 450 sectors, too large to fit on a single side of a cassette). Those with Diskette 1100 series computers do not have cassette tape drives and hence must use diskettes as their file transfer medium. Since DOS.C software is distributed on diskettes by Datapoint Corporation, the user will need to have at least a two drive system to copy the software from the diskette received from Datapoint to his working diskette(s). Single drive systems should be considered only by those intending to use them as satellite systems (example: data entry stations) and planning on having at least one other system with two or more drives (for program development and file processing applications).

Another option which should be strongly considered is the High Speed, or so-called "RAM" Display option for 2200 processors. This option can substantially increase total system throughput and responsiveness (especially in applications displaying a lot of text on the screen, such as data entry) at a very small additional cost. The RAM Display is field-installable (although less expensive when factory-installed), and is standard equipment on Datapoint 5500 and 6600 and Diskette 1100 series computers.

C.2 Performance of DOS.C

Users who are currently using Datapoint computers in cassette-based systems will find substantial improvements in performance when they upgrade to DOS.C. The 9380 series diskette drives are several times faster than cassettes for ordinary sequential data transfers; random-access type operations (such as sorting and ISAM file access) can easily be two orders of magnitude faster than is attained using tape cassettes.

Users who are currently using competitive diskette-based equipment will generally find that total system performance of Datapoint systems will exceed that which they are accustomed to. This improvement is due to the generally superior data handling techniques and file structuring as used in Datapoint's DOS. These characteristics stem from the fact that instead of employing a lower-performance cassette-style file structure as a base for the operating system, Datapoint chose instead to adapt the same advanced and dynamic disk file access techniques as used in its other DOS to the new diskette media.

The obvious side benefit of this DOS compatibility is that not only is virtually all of the Datapoint DOS software library available to diskette users but that most user programs which were written originally for other Datapoint DOS systems will run unmodified (except for possible file size limitations and timing differences due to the slower access times of the 9380 series disk drives) under DOS.C.

In addition to the increased speed of access of the 9380 series drives as compared to cassettes, another big advantage is that the total amount of storage available on a diskette-based system is about four times the amount usable on cassette systems, even when both cassette drives are in use.

It should be recognized that DOS.C is not expected to eliminate the usefulness of larger capacity, higher performance disks. Many users will have applications which are too involved or too large for the 9380 series diskette drives. Users who need large data files or high speed random access to disk storage will find the performance they are looking for in the other versions of Datapoint DOS.

C.3 Disk Drives

Datapoint DOS.C supports up to four 9380-series flexible diskette drives through their integral disk controller unit. The disk controller contains 1024 bytes of high speed, random access memory which buffers four sectors between the diskette drives and the Datapoint processor, enabling greater I/O device autonomy and improved overall system performance.

C.4 Disk Media

The Datapoint 9380-series flexible diskette drives use a flexible diskette for data storage. This diskette is media and format-compatible with the IBM 3741-style flexible diskette.

Data is recorded in 77 concentric circles on only one side of the diskette (as per the IBM standards for diskette data interchange). Each such circle is referred to as a track. Although each such track on the diskette actually contains 26 physical records of 128 bytes each, these are paired up by the Datapoint 9380 series diskette controller (an integral part of the diskette system) so that to the Datapoint computer each track appears to consist of 13 records (called sectors) of 256 bytes each. In Datapoint DOS documentation, unless explicitly indicated to the contrary, the term sector always refers to a 256-byte logical sector, and it is strictly incidental that this sector is broken into two physical 128 byte records for transfer to and from the diskette media.

The diskette is permanently enclosed within a durable plastic cover. This cover provides for easy insertion of the diskette into the diskette drives and provides structural rigidity for the media when it is not in use. In addition, the plastic cover provides a degree of environmental protection for the diskette and its oxide surface from damage caused by careless handling.

C.5 Loading and Unloading Diskettes

Upon observation of a diskette, three holes through the plastic diskette cover will be noted. Each of these holes allows one to see a portion of the oxide-coated surface of the diskette itself.

The large, round hole in the center of the cover is used by the diskette drive for the hub which clamps to the diskette and turns the diskette within the cover.

The longer, narrower radial slot towards one edge of the enclosure is the slot through which the read/write head in the diskette drive contacts the diskette's oxide coating for data transfer operations.

The smaller round hole present on the diskette is the hole through which the index hole, a hole in the diskette proper about the diameter of a pencil lead, is sensed by the controller and used for timing and control purposes.

The reason for the description of these holes is that they provide the definitive reference for indicating the proper direction of insertion of the diskette media into the 9380 series drives. When the diskette is properly inserted, the edge of the diskette with the long slot is inserted first. The smaller hole (the one through which the index hole is sensed) will be the last of the three holes in the cover to enter the drive, and it will be positioned toward the tabletop rather than downwards toward the floor.

The diskette loading slot is covered by a long, narrow handle. A rectangular pushbutton to the side of the handle is pushed to open the handle for diskette insertion and removal. When inserting the diskette, it will meet with a spring resistance after being inserted about 3/4 into the drive. Press the diskette gently into place until the spring catches and the diskette stays in place without being held in with the finger. Be careful not to push the diskette too far into the drive, as this can cause the innermost edge of the diskette's plastic cover to be wedged between some metal projections on the diskette drive which could possibly result in damage to the diskette. After the diskette is in place, pull the door/handle to the left until it latches closed. As the door is pulled closed, the hub engages the diskette, bringing it to its rated rotational speed of 360 rpm (and then online) almost immediately.

To remove a diskette, first ensure that all input/output activity on the diskette has completed. (This is necessary since it is possible to open the drive door, which takes the diskette offline, in the middle of a write operation; this can result in improper data being written onto the diskette.) Then press the button to the left of the door/handle. The door will open and the diskette will emerge in much the same way toast pops out of a toaster. Upon removing the diskette from the drive, it should be immediately placed in its protective paper envelope to help protect the surface from abrasive contaminants and other elements which could damage it.

C.6 Drive Numbering and Switches

Diskette drives are normally installed in the cabinet starting from the left. These drives are numbered 0, 1, 2, and 3 respectively from left to right. These numbers constitute the physical drive number. In the case of DOS.C, the same number is also sometimes referred to as the DOS logical drive number, or frequently just drive number.

The main power switch for the diskette unit is located on the underside of the tabletop and to the left side of the diskette drives, positioned toward the front of the cabinet. Sliding this switch towards the rear of the diskette drive cabinet turns the diskette unit on, and sliding the switch towards the user turns the diskette unit off. This switch should normally always be left in the ON position.

There are no other controls or switches intended for use by the user on the 9380-series diskette drives.

C.7 Care and Handling of Diskettes

Diskettes are sturdy media which will give long and trouble-free service if they are handled with reasonable care.

1) Diskettes should always be stored in their protective paper envelopes when not inserted in a drive. These envelopes should then be stored in the protective boxes in which the diskettes are received from the manufacturer.

2) Do not force too many diskettes into one box. They should not be placed under heavy pressure, as this can warp the diskette media, possibly causing read/write errors.

3) Diskettes should not be rolled, folded or otherwise subjected to strains which could crease the media.

4) Never touch the oxide coating of the diskette through the holes in the plastic cover. Human skin has oils on it which will attract and retain dust and other abrasive contaminants if these oils get onto the diskette's surface. In addition, contact between hard surfaces and the diskette oxide can scrape away the information-carrying oxide from the diskette surface; this will usually result in unrecoverable errors on the diskette.

5) Diskettes should not be subjected to strong magnetic fields.

6) When mailing diskettes, they should either be placed between two sheets of corrugated cardboard (for rigidity and protection while going through the mails) or placed in some suitable protective carrier. Many diskette media manufacturers sell mailers specifically designed for use in sending diskettes, either singly or in multiples, through the mail.

7) Diskettes can generally be taken through airport security x-ray and metal detecting equipment without danger of damage to the information recorded on the diskette.

C.8 Preparing Diskettes for Use

When a diskette is first received from the media manufacturer, it contains formatting information recorded across the entire usable surface of the diskette. This information is provided to allow the controller to identify where a given sector is on the surface of the disk, and also allows the controller to verify proper head positioning in the drive mechanism. Normal reading and writing on the diskette does not destroy the formatting information contained thereon.

Only diskettes in 3740 format (128 byte sectors) are usable by DOS.C. Diskettes that have been reformatted with bad tracks flagged and alternate tracks substituted cannot be used. Also, diskettes in System 32 format (256 byte sectors) or IBM 3600 format (512 byte sectors) cannot be used.

Diskettes cannot be used by DOS.C until they have been generated with the DOSGEN program described earlier. Datapoint DOS uses its own unique file structure which is capable of more sophisticated data and file manipulation than the standard IBM file structure which is intended for data entry and not for actual computer data processing. This more sophisticated file structure is what results in the need for DOSGEN before a diskette can be used by DOS.C.

A special note regarding disks which are to be used in the booted drive is appropriate. All of the newer releases of DOS commands use DOS FUNCTIONs (as described in the DOS USER'S GUIDE). These functions are resident on the diskette in the file SYSTEM7/SYS. When updated versions of DOS.C and associated utilities are received from Datapoint Corporation, the file

SYSTEM7/SYS may also have one or more new DOS FUNCTIONs resident. Therefore, wholesale copying of DOS commands from newly received diskettes to older diskettes with older versions of SYSTEM7/SYS will frequently result in commands which either work or do not work depending on whether the older or newer version of SYSTEM7/SYS is present on the booted drive. Therefore the user should generally keep his DOS commands disk more or less intact and not use a newly released diskette to supply commands to previously DOSGENed diskettes; instead, he should freshly generate as many system diskettes (including whichever DOS commands he intends to use) as he needs.

C.9 Suggested Disk Organization Techniques

Due to the relatively small capacity of the flexible diskette, careful consideration should be given to which files should be put on which diskettes. Users with single drives for data entry and related applications will have little choice in such matters. However, for users with multi-drive systems being used for program development, the following convention is suggested:

1) DOS system diskettes. These diskettes contain the system files (as do all diskettes for use with DOS.C) and whichever DOS commands the user intends to use. Usually this diskette will be used in drive zero during program generation, debugging and other DOS system-type functions and because of this will contain all of the DOS itself, DOS commands and latest set of DOS FUNCTIONs as released by Datapoint. This diskette will also frequently contain the editor scratch file, SCRATCH/TXT.

2) Source program diskettes. These diskettes can be considered as library file diskettes. These diskettes may contain programs, Dataform forms, and other user text files used, for example, during program generation. Once these programs are finalized, they can be copied to DOS System diskettes or User System diskettes as appropriate.

3) User system diskettes. These diskettes are similar in intent to DOS System diskettes but differ in that they are intended more for specific application use rather than for general program development and debugging. These diskettes will usually be used with DB11, SCRIBE, DOSBASIC, or DATAFORM or will have large numbers of user-written application programs on them. These disks will usually not contain the more specialized DOS commands

(and other files) such as DUMP9380, DSKCHECK, DOS/EPT, APP, CHANGE, DUMP and the like.

4) Data file diskettes. These diskettes contain primarily user data files. Typical characteristics of files on this type of diskette: non-executable, user information; SCRIBE text; other things which are user-entered (or generated) but not programs as such.

5) Scratch diskettes. These diskettes are diskettes containing no important files. These diskettes are suggested for use in transferring files from one diskette to another, and to provide diskettes containing large unallocated areas for use as scratch files by programs using scratch files (for example, SORT and EDIT commands).

As support for the above five basic types of diskettes, the following color-coding convention is suggested for diskette labels:

red - DOS System diskettes
green - User System diskettes
blue - Text files (source programs and SCRIBE text)
yellow - Data files
grey - Scratch diskettes

For best results, users should use only diskette media provided by those manufacturers recommended by Datapoint Corporation.

C.10 Disk Organization under DOS.C

This chapter describes the logical organization of the data on the disk when operating under DOS.C and how it relates to the general DOS file concepts as described in the chapter on System Structure. In this chapter it is assumed that the user is familiar with the basic DOS file structuring.

C.10.1 Radius Spiraling and Sector Skewing

Under DOS.C, the sectors on the diskette are logically renumbered to allow substantially increased performance over what would be possible otherwise. Program loading, in particular, goes about three times faster than would be possible if this were not done. This renumbering of the sectors on the track is referred to as sector skewing. This sector skewing takes the form of placing

logically sequential sectors about four sectors apart on a track of the diskette. Thus logical sector zero on track zero would appear in physical sector zero; logical sector one would appear in physical sector five; logical sector two would appear in physical sector ten; and so forth.

In addition to rearranging the order of the sectors on a track of the diskette, the starting points (logical sectors zero on each track) do not line up along a straight-line radius as do the physical sectors zero. Instead, the starting point for numbering sectors on a track spirals inwards. Therefore, the logical radius line (sectors zero, for example) forms a spiral on the diskette surface, and hence the term radius spiraling. The intention behind radius spiraling is twofold: one, it allows for head seek time between adjacent tracks while rapidly scanning through a data file (in addition to the processing time lag provided by the normal sector skewing); two, it allows searching the directory (which is along a logical radius of the diskette, as will be described later) about three times faster than would otherwise occur. Together with sector skewing, radius spiraling aids in achieving much higher overall system performance than is obtainable on most competitive diskette based systems.

Use the chart below to convert from the logical to physical sector. First divide the decimal track number by 4. The remainder gives the appropriate column. Run down the left side to the logical sector and across to the appropriate column to get the physical sector number.

LOGICAL SECTOR	REMAINDER OF TRACK/4			
	0	1	2	3
0 (0)	0	05	012	02
1 (01)	05	012	02	07
2 (02)	012	02	07	014
3 (03)	02	07	014	04
4 (04)	07	014	04	011
5 (05)	014	04	011	01
6 (06)	04	011	01	06
7 (07)	011	01	06	013
8 (010)	01	06	013	03
9 (011)	06	013	03	010
10 (012)	013	03	010	0
11 (013)	03	010	0	05
12 (014)	010	0	05	012

Note that physical sector addresses are never used by DOS. Even in DUMP9380 the sector address entered is taken as a logical

sector address except when in EBCDIC mode, when it is considered a physical sector address.

C.10.2 Size of a Diskette

There are 77 tracks on a diskette, each of which contains logically thirteen 256-byte sectors (physically twenty-six 128-byte sectors). This yields a total of 1001 sectors, or a grand total of 256,256 bytes of storage capacity. The first track on the diskette (the one nearest the edge of the diskette) is not used by DOS.C, in order to help provide compatibility with IBM equipment. Additionally, the logical last sector on each track (sector 12 if one counts starting at 0) is not used by DOS.C for data, for reasons which will be described in subsequent sections. Subtracting these two unallocatable areas results in a total allocatable file space of 912 sectors. About ninety sectors of these are used by the DOS for its system files, leaving about 825 sectors for user files, a user file capacity of over 200,000 bytes. This constitutes about twice the capacity of a tape cassette on each diskette. Due to the higher data storage efficiency attained by Datapoint software, most users will find that the total number of records stored on a Datapoint format diskette will be as large as, and in most cases substantially larger than, the number achieved on competitive systems.

C.10.3 Cluster Mapping

Under DOS.C, each track of the diskette consists of 4 clusters of three sectors each. This implies that one cluster or three sectors is the smallest allocatable unit of space on a diskette, and that all files are multiples of three sectors in length.

In the cluster allocation table, the four clusters on each track are represented by the low-order four bits of each byte. As in other Datapoint DOS, a one bit represents that the corresponding cluster is allocated and a zero bit indicates that the corresponding cluster is available for allocation. The high-order four bits of each byte in the CATs are reserved for future use in DOS.C, and are currently always set to zero.

C.10.4 Segments under DOS.C

The use of a three sector cluster has numerous advantages on the diskette. One which should be immediately apparent is that the amount of space wasted due to always allocating an integral number of clusters is reduced to only an average of one and a half sectors per file. Perhaps a less obvious advantage results from the manner in which the Datapoint DOS allocates disk space to files. During space allocation, the DOS will allocate the first contiguous, maximum-size segment it can find as an initial (or secondary) allocation. Since a segment consists of up to 32 clusters (there are five bits of cluster number information in each segment descriptor), this results in files being initially allocated 96 sectors, assuming that the space on the diskette is sufficiently unfragmented to allow such an allocation. Making this initial allocation smaller than the 192 and 240 sectors as used in some of the other Datapoint DOS allows for several scratch files to be opened on a diskette which already has a few files on it, as each newly opened file will take a smaller bite out of that portion of space remaining unallocated. Making the full segment size much smaller than 96 sectors quickly increases the amount of overhead required to index through the file (since the number of RIB accesses required increases) and decreases performance.

C.10.5 Maximum File Size

Under DOS.C, the maximum file size attainable depends upon the amount of space used on the diskette for system files, but using the current size of DOS.C as an example indicates that at least 800 sectors should be available for user file allocation on a normal data file diskette. Users having only a single diskette drive and therefore having several programs on the diskette in addition to the DOS will have a corresponding reduction in the maximum size of data files they may have. Users whose files exceed the capacity of one diskette will need to segment their files at the user program level much as they would do on a cassette system when a file exceeded the capacity of a single cassette.

C.10.6 Cluster Allocation Table and Directory

Under DOS.C, the use of four three-sector clusters per track results in one unused sector per track. This restriction arises from the facts that (1) all clusters under Datapoint DOS must contain the same number of sectors and no cluster may span a track boundary; and (2) a 13-sector cluster is not practical because it results in excessive amounts of wasted space at the end of each file on the diskette. Since these 76 sectors on the diskette (remember that track zero is not used) are not available for allocation as file space, they are partially put to use for storage of DOS system tables: four cluster allocation table sectors and thirty-two directory sectors. These system tables are positioned in the following manner:

```
Track      0 - Unused;  for IBM compatibility
Tracks  1-16 - Directory copy, for backup purposes
Tracks 17-32 - Primary DOS directory
Track     33 - Working Cluster Allocation Table
Track     34 - Working Cluster Allocation Table backup
Track     35 - Lockout Cluster Allocation Table
Track     36 - Lockout Cluster Allocation Table backup
Tracks 37-76 - Reserved for future DOS use
```

Again recall that each of the above sectors is in logical sector 12 of the track indicated.

In the Cluster Allocation Tables, bytes 239-254 are used for the Directory Mapping bytes. These sixteen bytes each contain either an 0377 or the number of files currently allocated in the corresponding one of the sixteen directory sectors. These bytes are updated automatically by the DOS whenever a file is created or deleted, and are updated by the DOS occasionally if they are found to be inaccurate. The purpose of these directory mapping bytes is to provide improved speed of directory lookups and to allow faster creation of files. They are of the greatest benefit to users who have several drives in their system where relatively few files exist on each drive. The intention is to eliminate the need to read in directory sectors while looking for a file if those sectors are known to not contain any active directory entries, and likewise when looking for an empty slot for use by a new file to eliminate having to read sectors known to have all sixteen directory entries in use.

C.11 Internal DOS Parameterization

This section describes the DOS-dependent details of the parameterization of DOS.C system routines.

C.11.1 Physical Disk Address Format

Under DOS.C, physical disk addresses are represented in a two-byte format in a manner quite similar to that used under the other DOS. The most significant byte (which is traditionally placed in the D register) is the cylinder number. The less significant byte (usually placed in the E register) has its most significant two bits representing a cluster number within the track (all combinations of these two bits are valid since there are four clusters per track) and the least significant two bits representing the sector number within the specified cluster. Because there are only three sectors per cluster, only binary values 00, 01 and 10 are valid for these low-order bits. (The only exception to this rule is that a least significant PDA byte of 0303 permits access to the unallocatable sector on each track, that sector used for system table sectors). (For compatibility reasons, the most significant three bits can be considered the cluster number, yielding clusters numbered 0, 2, 4, and 6).

The unused bits of the least significant physical disk address byte (that is, the center four bits) should always be set to zero.

APPENDIX D. DOS.D

DOS.D is Datapoint Corporation's Disk Operating System supporting 48K or larger Datapoint 5500 or 6600 family computers operating in conjunction with from two to eight 9370 series disk drives, or with from two to three 9390 series disk drives. In addition to the interactive/batch operation as provided in all standard Datapoint Corporation DOS, DOS.D additionally supports Datapoint's Partition Supervisor (called PS, PS66, or UPS, released separately) which provides for the simultaneous execution of multiple DOS programs. DOS.D is also the operating system used in the Datapoint Attached Resource Computer (ARC) system.

D.1 Planning for DOS.D

The minimum configuration for a DOS.D system requires a 48K Datapoint 5500 family computer. The minimum disk requirements are a 9370 controller and drive and at least one 9371 disk extension unit, or a 9374 controller and drive, or a 9390 controller and dual drive. If more storage is desired, additional disk extension units may be attached to the disk controller, up to a total capacity of 16 logical disk drives.

D.2 File Storage Capacity under DOS.D

Under DOS.D, each 9370-9373 model disk unit is dealt with as two logical drives. Each of these two logical drives contains 48,576 sectors of 256 bytes each and can store up to 256 files. Of these, about 250 sectors and about ten files are used by the operating system and a few basic commands, leaving about 12.4 million bytes of usable space per logical drive, or up to roughly 25 million bytes of storage total for each disk storage unit in the configuration.

Using the 9374/9375 disk drives, file storage is somewhat less due to the capacity of the disks used. For these drives, each logical drive contains 38,976 sectors and provides about 10 million bytes of usable space per logical drive, or roughly 20 million bytes of storage per disk unit.

Using the 9390 series disk drives, each physical disk pack is handled as 5 logical drives, each logical drive being identical to a logical drive on a 9370 disk pack. Thus each logical drive has

48,576 sectors, providing about 12.4 million bytes of usable space. Each physical disk unit provides about 62 million bytes of storage.

Other features of DOS.D include a large maximum file size: up to 38,397 data sectors in a single DOS.D file (not including the end-of-file mark and two RIBs).

D.3 Disk Drives

Datapoint DOS.D supports from two to eight 9370-series disk drives attached to one 9370-series disk controller, or one 9390 controller and dual disk drive unit and optional extension drive. These drives are high-performance, random access disk units. They are the equal in every way to drives in constant daily use on the largest mainframe computer systems.

D.4 Disk Media

See Appendix B for information on the 9370-series disk drives and disk packs.

The Datapoint 9390 series disk drives use a 5-platter disk pack with six recording surfaces, one of which is used only for timing marks. These packs are equivalent to CDC Model 877 packs. The disk pack is enclosed within a plastic enclosure when it is not in place in the drive. This cover is intended to help keep the disk free from dust, pollen, smoke, and other contaminants and is of prime importance in helping to eliminate disk errors and parity failures that contamination can cause. Keep the disk in its cover at all times when not in use.

D.5 Loading and Unloading Disk Packs

See Appendix B.5 for information on the 9370 series disk drives and disk packs.

The model 9390 drives are top-loading units. To open a drive, release the lid by lifting up on the latch at the center of the front of the drive, just under the lip of the lid. The lid will lift open, exposing the drive cavity. The hinges of the lid are damped so it will remain open when released. The lower drive of the pair in a cabinet must be slid forward out of the cabinet before its lid can be opened. To slide the lower drive forward, use the depressions in the lower front of the drive. Push up

against the latch at the top of either of the depressions until the drive is released, then pull gently forward until the drive is clear of the cabinet.

To insert a disk pack, hold the pack by the top center handle. Release the bottom cover of the pack by squeezing the two halves of the small handle in the center of the bottom cover toward each other. Remove the bottom cover. Still holding the pack by the top center handle, lower the pack and top cover together into the drive cavity. When the pack is seated on the drive spindle, turn the top center handle fully clockwise until firm resistance is met. It is important that the pack be firmly in place before removing the top cover. When the pack is properly mounted, carefully remove the top cover by lifting it straight up.

Whenever moving the top cover in or out of the drive cavity, be sure to move the cover carefully straight up or down. If the cover becomes skewed in the cavity, damage to the disk pack or to the drive could result.

Once the pack is in place, close the drive access lid, which will latch into place. The lower drive should be immediately pushed gently back into the cabinet until it latches into position. The top and bottom halves of the disk pack cover should be immediately put back together to keep out dust and other contaminants.

To remove a disk pack, first stop the drive (see Switches and Indicators below) and be sure the drive is in load condition. Open the drive access lid. Carefully lower the top cover straight down onto the disk pack. When the cover is fully lowered, turn the top center handle counterclockwise until a distinct click is heard. This click indicates that the pack has been released from the drive spindle and may now be removed. Lift the disk pack and top cover together straight out of the drive and immediately reattach the bottom pack cover by putting it in place and pressing its latches into position. The pack should be stored horizontally on a shelf (never on edge!) and in a position where it is not apt to be dropped or pushed accidentally over an edge. If another disk is not to be mounted immediately into the drive the pack was just removed from, the disk access lid should be closed and the drive pushed back into the cabinet right away to avoid possible physical damage or contamination.

D.6 Switches and Indicators

See Appendix B.6 for information describing the 9370 series disk drives.

The controls of the 9390 disk drive are in a small recessed panel in the upper left hand corner of the front of the drive unit. The physical drive number indicator is located below the READY light on the panel. This indicator is the front of a removable plastic plug that is keyed to select the physical drive number of the unit. DOS can use physical drive 0, 1, and 2 only.

When the drive is in load condition, both the START and READY lights will be off. To start the drive, push the button below the START light. The START light will immediately come on; the READY light will blink while the disk pack is spun up to speed. When the drive is ready for operation, the READY light will stop blinking and stay lit. During operation, both the START and READY lights will stay lit. To stop the drive, again push the button below the START light. The START light will immediately go off; the READY light will blink while the disk pack is braked to a stop. When the drive is again in load condition, both the START and READY lights will be off.

The READY light is the main indicator of drive status. When the READY light is off, the drive is in load condition and disk packs may be loaded or unloaded. When the READY light is blinking, the drive is starting or stopping and no operations are possible. The 9390 takes less than 30 seconds to become ready for operation or to stop following operation. When the READY light is lit, the drive is ready for I/O operations.

The PROTECT light indicates the write-protect condition of the drive. When the PROTECT light is lit, the disk pack in the drive is physically protected and cannot be written upon. When the PROTECT light is off it is possible to write on the disk. Write protection is controlled by the pushbutton beneath the PROTECT light. If protection is off, pushing the button turns protection on; if protection if on, pushing the button turns protection off. The button extends farther from the cabinet while protection is off, but the difference in position between the two conditions is difficult to see.

The FAULT light comes on when the safety circuits in the drive detect one or more of a number of conditions that they consider would endanger the data on the disk pack if continued disk operation were attempted. The FAULT condition can be caused by (among other things) power surges or power supply problems in

the drive, or by illegal commands from the controller. The FAULT light can be cleared by pressing the button below it. If the light stays off, the error condition causing the fault has been corrected and operation can continue. If the light comes back on immediately, the condition still exists and further work must be done to clear it. The drive should be stopped, then brought ready again, and the DOS should be restarted. If the fault condition still persists, or if a transient fault keeps recurring, a hardware malfunction is indicated and the problem should be reported to the Datapoint Customer Support Center.

D.7 Disk Organization under DOS.D

This section describes the logical organization of the data on the disk when operating under DOS.D and how it relates to the general DOS file concepts as described in the chapter on System Structure. In this section it is assumed that the user is familiar with these concepts and has read and is familiar with the basic DOS file structuring.

D.7.1 Logical Drive Mapping

Under DOS.D each physical drive is broken into multiple logical volumes. This is done for reasons of addressing. It is simply not possible to address all of the sectors on an entire mass storage disk drive using only two bytes of physical disk address, and the two byte physical disk address is central to all of Datapoint Corporation DOS's operations. It is not practical to change this characteristic without making changes which would result in invalidating many user-written programs and many large systems which run under the DOS. Therefore the disk was broken into logical volumes, and part of the effective physical disk address is taken from the logical drive number.

For the 9374 drives the removable disk is one logical drive, and the fixed disk is a second logical drive.

For the 9370 drives, each disk pack is two logical drives. Each logical drive appears to be 253 cylinders (numbered 0-252 decimal) of eight 24-sector tracks each. The first eight recording surfaces on the disk pack (heads on the 9370 drive are numbered from zero to nineteen starting at the top of the disk drive) correspond to the first 203 cylinders on the first logical drive (the even-numbered one). The next eight recording surfaces on the disk pack correspond to the first 203 cylinders on the second logical drive (the odd-numbered one). The first 203

cylinders on each logical drive is referred to as primary addressing space. Mapping of disk space within primary addressing space is done in an algorithm identical to that used under DOS.B.

Physical heads sixteen and seventeen (and the corresponding recording surfaces on the disk pack) correspond to logical cylinders 203-252 on the even logical drive; heads eighteen and nineteen correspond to logical cylinders 203-252 on the odd logical drive. These cylinders of each logical drive are referred to as the extended addressing space. Since DOS.D assumes that each cylinder consists of eight tracks, each of logical cylinders 203 through 252 are mapped across four physical cylinders of two tracks each from the center of the pack outward. In this way, disk space within primary and extended addressing space can be dealt with by DOS.D in a uniform way at all but the very lowest levels of the disk read/write driver.

Using the 9374/9375 disk drive, there is no extended addressing space, only 203 cylinders of 8 24-sector tracks each. The disk platter itself has 406 cylinders of 2 48-sector tracks each, but the disk controller provides address conversion so the physical structure is transparent to the processor.

For the 9390 drive, each disk pack is addressed as five logical drives. Each logical drive is addressed in the same manner as a logical drive on the 9370 disk. The mapping of the physical disk structure to a logical structure of 253 cylinders of 8 tracks each with 24 sectors per track is handled by the 9390 controller. The actual physical structure of the disk cannot be addressed by DOS, so only the logical structure presented by the controller is considered in this manual.

D.7.2 Size of a Logical Drive

D.7.2.1 Models 9370-9373 and 9390-9393

Each logical drive is eight tracks on each of 253 cylinders. This results in 2024 tracks of 24 sectors each, or a total of 48,576 total sectors on a logical drive. Of these, about 48,000 remain when the DOS has been generated, the system tables constructed on the disk, and a few basic commands loaded.

D.7.2.2 Models 9374/9375

Each logical drive is eight tracks on each of 203 cylinders, yielding 1624 tracks of 24 sectors each, or 38,976 sectors on a drive. Of these total sectors, about 38,400 remain when the DOS has been generated, the system tables constructed, and a few basic commands loaded. There is one unused cylinder on each platter -- logical cylinder 203, physically cylinders 406 and 407. These innermost cylinders are not normally addressable and are not even formatted by INITDISK. A test program for long-term reliability testing is planned which will require exclusive use of these cylinders.

D.7.3 Cluster Mapping

Because there are eight bits per byte in the cluster allocation table (or CAT for short), and it is desirable to maintain one byte in the CAT per cylinder of available space on the drive, each track on a logical drive represents one DOS cluster, and is represented in the CAT by exactly one bit. Since the DOS uses eight tracks per logical cylinder, this results in exactly eight clusters per cylinder of twenty four sectors each.

Due to the fact that space is always allocated in terms of an integral number of clusters, this implies that the minimum file size under DOS.D is twenty-four sectors and that file size will always be a multiple of this number. It turns out that choosing a full track as the smallest allocatable unit of space has other advantages as well from a system standpoint, since it allows some programs (like COPY) to make several simplifying assumptions about the data in a file which enables them to copy data and reference information in a file substantially more readily and efficiently than would be otherwise possible.

D.7.4 Segments under DOS.D

Space under Datapoint Corporation's DOS is always allocated in contiguous chunks of clusters called segments. When space is allocated, the largest segment on a drive up to the maximum possible sized segment is allocated, to keep the file as free of fragmentation as possible. By limiting the allocation size to the size of a full segment, the problem of allocating all available space on a disk to a first scratch file before a second one is subsequently opened is minimized. If several scratch files are opened and space in them is allocated at regular intervals, the resultant segments will be interleaved, resulting in minimized

access time as the heads randomly access throughout the scratch area. The desire to make segment size small (to minimize file space conflicts and help to optimize use of space on the drive) and yet large (to maximize processing speed, maximize file size and minimize the number of RIB accesses) resulted in a segment size of thirty-two clusters. This compromise results in a 768-sector segment (32 clusters of eight tracks of 24 sectors each) allowing a maximum file size of over 38,000 sectors.

D.7.5 Maximum File Size

Under DOS.D there is no fixed upper limit on file size. A file may be as large as will fit in the remaining space on a volume. On a 9390 or 9370 this works out ot be about 48,000 sectors. On a 9374 this works out to be about 38,400. Files longer than these numbers will have to be segmented or logically concatenated at the user program level, the same as would be necessary for files larger than about 9600 sectors on the 9350 series disks.

D.7.6 Cluster Allocation Table and Directory

Each logical drive under DOS.D contains its own directory and cluster allocation table, just as for all Datapoint Corporation DOS. There are sixteen directory sectors on each logical drive, located in consecutive sectors starting at sector seven on logical track zero of cylinder zero. Therefore, the sectors go from sector seven to sector 026 (octal). The cluster allocation table is at sector zero of logical track zero, cylinder zero. The lockout cluster allocation table is at sector one of logical track zero, cylinder zero. The hashed directory index is at sector two of track zero, cylinder zero. The backup copies of each of these are in the corresponding locations of logical track one of the same cylinder.

The Hashed Directory Index, maintained by the DOS, resides in sector two of track zero, cylinder zero. This table enables directory lookups to go about four times faster (on full disk directories) than was possible under DOS.B Version 1. The technique works as follows:

Given an eleven byte file name and extension, an arithmetic/logical operation upon the file name results in an eight-bit quantity referred to as a hash code. This code is essentially a condensation of the 11 bytes of file name and extension information into only one byte. Obviously, the

information is not complete; there are only 256 distinct eight-bit hash codes possible, while there are literally billions of legal file names and extensions. However, the condensation of information is such that looking at the hashed directory index allows the DOS to substantially restrict the range of directory sectors it must examine when doing a directory lookup. Each hash code for the file names in the directory is stored into the hashed directory index, offset by the physical file number (PFN) of the file with the corresponding name and extension.

Note that there is a calculated danger in the hashed directory approach. The danger is that if the hashed directory index is overwritten or otherwise destroyed accidentally, files may become inaccessable even though they are clearly shown (by doing a CAT command on the disk, for example) to be present. When this occurs on a disk the technique to repair the disk is the DSKCHECK command. When the DSKCHECK command is almost finished, simply specify that the Hashed Directory Index is to be rewritten to the disk. This causes the HDI to be regenerated from the actual disk directory and rewritten. In general, the Hashed Directory Index is rarely if ever destroyed in actual disk usage, and contributes greatly to overall system performance if many directory lookups are being done.

D.8 Internal DOS Parameterization

This section describes the DOS-dependent details of the parameterization of DOS.D system routines.

D.8.1 Physical Disk Address Format

Under DOS.D physical disk addresses are represented in a two-byte format in a manner quite similar to that used under the other DOS. The most significant byte (which is traditionally placed in the D register) is the cylinder number. The less significant byte (usually placed in the E register) has its most significant three bits representing a cluster number within the cylinder (or logical track number in the specific case of DOS.D) and the least significant five bits representing the sector number within the specified cluster.

APPENDIX E. DOS.G - 1800 OPERATING SYSTEM

DOS.G is the Datapoint 1800 Disk Operating System supporting the 1800 processor with integral double-density diskette drives.

The 1800 processor's instruction set is the same as that of a 5500, as is its I/O structure. The most salient architectural differences are the direct coupling of the 1800 console screen to memory, and the 24x80 display.

The standard DOS commands are available, with the exceptions of these dependent on peripherals not supported by the processor (no cassette drives are included with the 1800 processor).

The following commands are not supported because they are associated with peripheral devices not available with an 1800 processor:

DUMP93X0
INITDISK
MIN
MOUT
REWIND
UBOOT

This Appendix describes the relationship of DOS.G to the other DOSs. DOS.G is designed to execute on a Datapoint 1800 processor connected to at least one dual-drive, double-density diskette. DOS.G supports a maximum of eight (8) logical diskette drives (4 physical drives) for a total of about 4 megabytes of on-line data. Each physical diskette drive can contain two (2) logical drives. Diskette organization and CRT/keyboard differences should have minimal impact on the user.

E.1 CRT / Keyboard Interface Under DOS.G

The CRT (screen) / Keyboard on the 1800 processor have several major differences from the 2200/5500 type processors:

1. 24 lines (vertical) instead of 12 lines.
2. Characters on the CRT are displayed from RAM memory.
3. Keyed-in characters and function keys are stored in RAM memory.
4. Additional "Function" keys have been added.

E.1.1 Screen Line Numbering

The 1800 CRT has 24 lines of 80 characters each. The lines are actually numbered starting at the bottom of the screen (for compatibility reasons). The bottom line number is eleven (11), as on the 2200/5500 CRT. Each higher line is numerically one less. This means that the top line number is minus twelve (-12).

ILLUSTRATION 1. CRT LINE NUMBERING

```
LINE -12 (Top of Screen)
LINE -11
LINE -10
LINE -9
LINE -8
LINE -7
LINE -6
LINE -5
LINE -4
LINE -3
LINE -2
LINE -1
LINE 0   This line corresponds to the top of a 2200/5500 CRT
LINE 1
LINE 2
LINE 3
LINE 4
LINE 5
LINE 6
LINE 7
LINE 8
LINE 9
LINE 10
LINE 11   This line corresponds to the bottom of a 2200/5500 CRT
```

The correct technique for determining the top and bottom line for any program that runs under DOS.G version 2.3 (or upward) is to call DOS Function 6, subfunction 4 (homeup position) and subfunction 5 (homedown position).

E.1.2 Displaying on the Screen

Displaying data on the screen can be accomplished in several ways:

1. DSPLY$ - This standard DOS routine is totally compatible with the pre-existing DSPLY$ routine.

2. DOS Function 6, subfunction 3 - write a byte to screen. This routine will write the byte in the (B) register to the screen, at the position defined by the (DE) register pair; D contains the horizontal screen coordinates, and E the vertical.

3. Writing a byte into the RAM memory that is the screen buffer. This technique is not recommended.

4. 1800 ROM display routine as documented in the 1800 Product Specification. This technique is not recommended.

E.1.3 Inputting Data From the Keyboard

Entering data from the keyboard can be accomplished several ways:

1. KEYIN$ - This standard DOS routine is totally compatible with the pre-existing KEYIN$ routine.

2. DOS Function 6, subfunction 2 - input a character. Register pair (DE) must contain the horizontal and vertical screen coordinates for the flashing cursor.

3. Reading the RAM byte in memory that is the keyed in character. This technique is not recommended.

4. 1800 ROM keyin routine as documented in the 1800 Product Specification. This technique is not recommended.

E.1.4 Special CRT / Keyboard Features

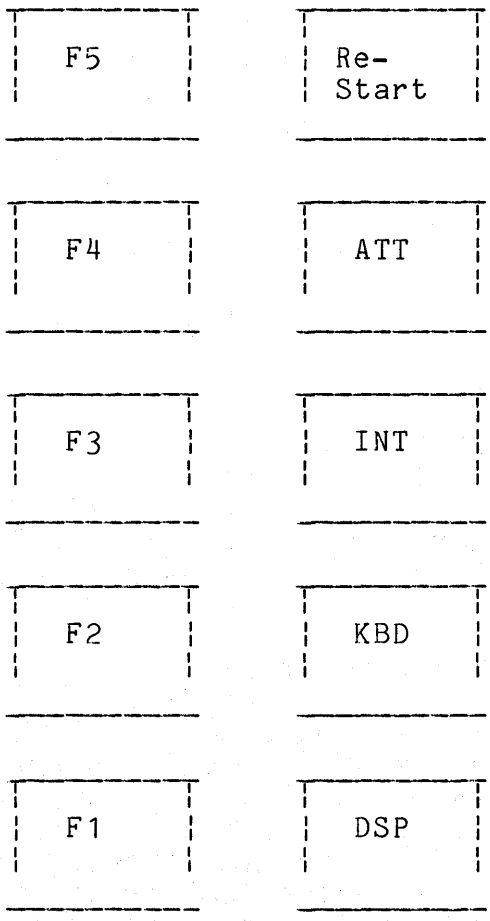
The 1800 CRT / Keyboard contains several useful features:

1. Inverted video capability. Each character position on the screen is defined as a 7x9 dot matrix. Normally, the inner 5x7 dots are used to display a lighted character. However, by displaying a character whose "sign" bit is set, the character will be displayed as a "hole" surrounded by "light". For example, writing an octal

"0101" to the screen will display the familiar character "A", however, displaying an octal "0301" will display a lit 7x9 rectangle containing a dark 5X7 outline of the character "A".

2. Additional "Function" keys. There are ten keys on the right-hand side of the 1800 processor. These are "function" keys. When they are depressed, the status bits change to reflect the fact that a key has been pressed. Each key sets a unique bit, therefore, any arbitrary meaning can be attached to any key (or combination of keys). The only word of caution necessary is that pressing the RESTART and INT keys simultaneously will cause the 1800 to perform a RESTART. Additionally, the KBD and DSP keys are treated like the 2200/5500 "KEYBOARD" and "DISPLAY" keys by DOS function 6.

ILLUSTRATION 2. FUNCTION KEYS



3. Roll-Down capability. The 1800 screen can "roll down", just like the 5500 screen (and 2200's equipped with RAM screen option). To roll the screen down, use DOS function 6, subfunction 10.

4. Character Font loading capability. Any combination of bits that can be represented in a 5x7 pattern can be displayed as a character. See the 1800 hardware reference manual for a description of character font loading. For simple non-standard characters, a "CHARSET/SYS" file may be used, as described in the manual for CHARIN18 (DOS.G International Character Set Loader).

E.2 Diskette Organization Under DOS.G

This section describes the logical organization of the data on the diskette when operating under DOS.G and how it relates to the general DOS file concepts as described in the chapter on System Structure. In this chapter it is assumed that the user is familiar with the basic DOS file structuring.

E.2.1 Loading and Unloading Diskettes

The 1800 diskette drives are aligned in a horizontal plane, rather than in a vertical plane (9380-diskette controller). This makes loading and unloading the diskettes much easier. Each diskette drive has two long handles (horizontal), and two smaller rectangular "unload" buttons. The unload button should be firmly pressed and released to open the diskette loading slot. The diskette should be carefully inserted into the 1800 diskette drive, "label side up", with the edge of the diskette that has the long, narrow slot being inserted first. When inserting the diskette, it will meet with a spring resistance after being inserted about 3/4 into the drive. Press the diskette gently into place until the spring catches, and the diskette stays in place without being held in with the fingers. Be careful not to push the diskette too far into the drive, as this could cause damage to the diskette media. After the diskette is in place, pull the door/handle straight down until it latches closed. As the door is pulled closed, the hub engages the diskette, bringing it to its rated rotational speed of 360 rpm, and then online, almost immediately.

To remove a diskette, first ensure that all input/output activity on the diskette has completed. Generally, the red indicator lights on the unload buttons will be a good indication of activity. Now, press the unload button firmly. The door will open and the diskette will emerge. Upon removing the diskette from the drive, it should be immediately placed in its protective paper envelope to help protect the surface from abrasive contaminants and other elements which could damage it. Also, the spinning hub of the drive itself could abrade the recording surface of a diskette left in a half-in, half-out position.

E.2.2 Drive Numbering

Each 1800 diskette drive contains two diskettes. The diskette on the right is the even numbered drive, and the one on the left is the odd numbered drive.

E.2.3 Care and Handling of Diskettes

Please read Appendix C, section 7 for the proper care and handling of diskettes.

E.2.4 Preparing Diskettes for Use

DOS.G supports double-density diskettes. This means that each diskette can contain about a half-megabyte of data. Before a diskette can be used, it must be "formatted" for double density mode. Executing the DOS. BACKUP command, or the DOS. DOSGEN command will allow the user to quickly convert a diskette to double density mode. A diskette that has been converted to double density mode is readable only on an 1800-type diskette drive.

E.2.5 Sector Skewing

Under DOS.G the sectors on the diskette are logically renumbered to allow substantially increased performance over what would be possible otherwise. This renumbering of sectors on the physical track, to build a logical track, is referred to as Sector Skewing. This sector skewing takes the form of placing logically sequential sectors three sectors apart on a track of the diskette. Thus, logical sector zero on track zero would appear in physical sector zero; logical sector one on track zero would appear in physical sector three; and so forth.

Since a logical track occupies the same space as the corresponding physical track , the difference being in the numbering of the included sectors, the terms may be used with some interchangeability.

E.2.6 Size of a Diskette

There are 77 tracks on a diskette, each of which contains 26 sectors of 256 bytes. This yields a total of 2002 sectors, or a grand total of 512,512 bytes of storage capacity. The first track (track 0) is used by the DOS for the IPL and BOOT blocks. Additionally, logical sector 24 (starting at 0) of each track is not used by DOS.G for data, for reasons which will be described in the Cluster Mapping paragraph. Subtracting these two unallocatable areas results in a total allocatable file space of 1900 sectors. About 90 of these sectors are used by the DOS for its system files, leaving about 1800 sectors for user files, a user file capacity of over 460,000 bytes.

E.2.7 Cluster Mapping

Under DOS.G, each track of the diskette consists of 8 clusters of three sectors each. This implies that one cluster or three sectors is the smallest allocatable unit of space on a diskette, and that all files are multiples of three sectors in length. Each byte in the cluster allocation table (CAT) represents a track, and each bit of a CAT byte represents a cluster (three sectors). Sector 24 of each track is used by the DOS. Sector 25 is unused by the DOS.

E.2.8 Segments Under DOS.G

The use of a three sector cluster has numerous advantages on the diskette. One which should be immediately apparent is that the amount of space wasted due to always allocating an integral number of clusters is reduced to only an average of one and a half sectors per file. During space allocation, the DOS will allocate the first contiguous, maximum-size segment it can find as an initial or secondary allocation. Since a segment consists of up to 32 clusters (there are 5 bits of cluster number information in each segment descriptor), this results in files being initially allocated 96 sectors, assuming that the space on the diskette is sufficiently unfragmented to allow such an allocation. Making this initial allocation smaller than the 192 or 240 sectors as used in some of the other Datapoint DOS's allows for several

scratch files to be opened on a diskette which already has a few files on it, as each newly opened file will take a smaller bite out of that portion of space remaining unallocated. Making the full segment size much smaller than 96 sectors quickly increases the amount of overhead required to index through the file (since the number of RIB accesses required increases) and decreases performance.

E.2.9 Cluster Allocation Table and Directory

Under DOS.G, the use of eight three-sector clusters per track results in two unused sectors per track. This restriction arrises from the facts that (1) all clusters must contain the same number of sectors and no cluster may span a track boundary; and (2) a 26 sector cluster is not practical because it results in excessive amounts of wasted space at the end of each file on the diskette.

Half of the 76 unused sectors, one per track, are put to use for storage of DOS system tables (four cluster allocation table sectors and thirty-two directory sectors). These system tables are positioned in the following manner:

ILLUSTRATION 3 - SYSTEM TABLE POSITIONS

Track	0	- Unused
Tracks	1-16	- Directory Copy
Tracks	17-32	- Primary DOS Directory
Track	33	- Working C.A.T.
Track	34	- Working C.A.T. Backup
Track	35	- Lockout C.A.T.
Track	36	- Lockout C.A.T. Backup
Tracks	37-76	- Reserved for future DOS use

Each of the above sectors is in logical sector 24 of the track indicated.

In the Cluster Allocation Tables, bytes 239-254 are used for the Directory Mapping Bytes. These sixteen bytes each contain the number of files currently allocated in the corresponding one of the sixteen directory sectors. These bytes are updated automatically by the DOS whenever a file is created or deleted, and are updated by DOS occasionally if they are found to be inaccurate. The purpose of these directory mapping bytes is to provide improved speed of directory lookups and to allow faster creation of files. They are of the greatest benefit to users who have several drives in their system, where relatively few files exist on each drive. The intention is to eliminate the need to

read in directory sectors while looking for a file, if those sectors are known not to contain any active directory entries; and likewise, when looking for an empty slot for use by a new file, to eliminate having to read sectors known to have all sixteen directory entries in use.

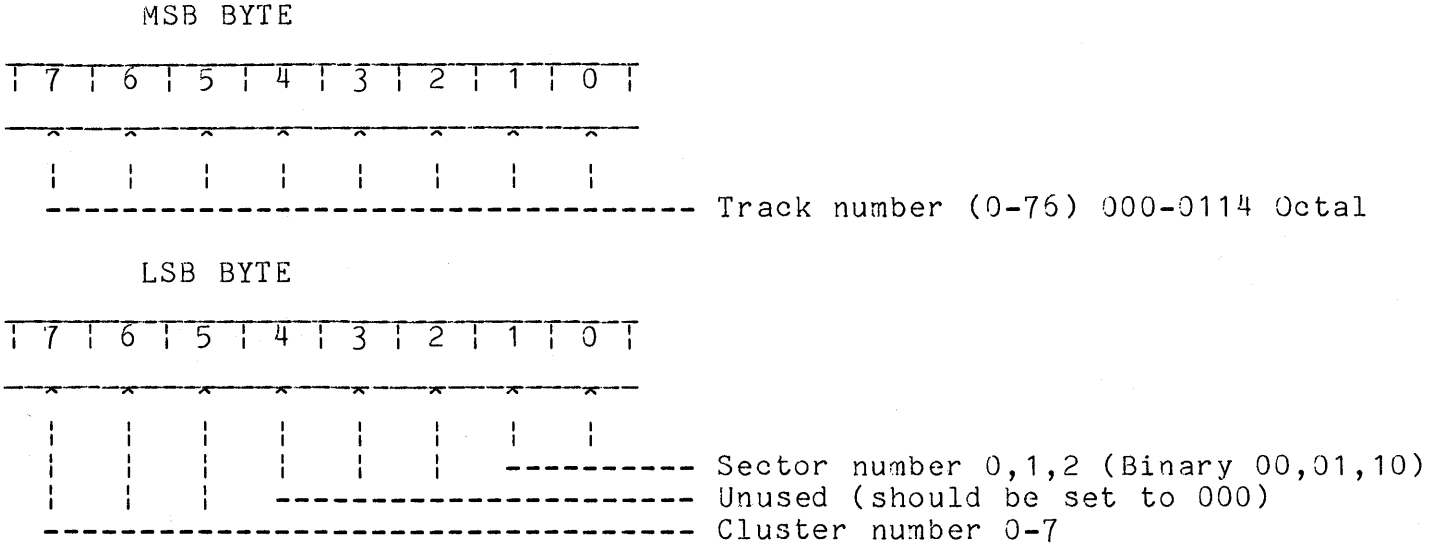
E.3 Internal DOS Parameterization

This section describes the DOS-dependent details of the parameterization of DOS.G system routines.

E.3.1 Physical Disk Address Format

Under DOS.G, physical disk addresses are represented in a two-byte format in a manner quite similar to that used under the other DOS. The most significant byte is the track number. The least significant byte has its most significant three bits representing a cluster number within the track and the least significant two bits representing the sector number within the specified cluster. Because there are only three sectors per cluster, only binary values 00, 01, and 10 are valid for these low-order bits. The unused bits of the least significant physical disk address byte should always be set to zero.

ILLUSTRATION 4 - P.D.A. FORMAT



APPENDIX F. COMPARSION CHART FOR DOS'S

The tables below list basic logical and physical configuration information for the various Datapoint DOS. When values refer to "drive" a logical drive is indicated, unless the value is specifically identified as referring to a physical drive. The information shown for DOS.B and DOS.D uses the following convention to distinguish among the various drive types supported by those systems: values for 9370 drives have no special punctuation, values for 9374 drives are enclosed in parentheses, values for 9390 drives are enclosed in square brackets. When only a single value is given for DOS.B or DOS.D, the information applies to all drive types.

	DOS.A	DOS.B	DOS.C
min processor req.	16K 2200	16K 2200	16K 1100
disk controller/drive type used	9350 9354	9370 (9374)	9380
phys. drives on system	1-4	1-2	1-4
logical drives per physical drive	1	2	1
type of disk	Scotch 92-204 or equivalent	Scotch 911-0 or equivalent (Datapoint model 80428)	IBM 128-byte soft-sectored diskette, or equivalent
cylinders used/drive	203	203	76
tracks used/drive	406	1,624	76
sectors/track	24	24	13
sectors/drive	9,744	38,976	988
bytes/drive	2,494,464	9,977,856	252,928
user sectors/drive	9,600	38,400	800
user bytes/drive	2,457,600	9,830,400	204,800
sectors/cluster	6	24	3
clusters/track	4	1	4
clusters/cylinder	8	8	4
max.clusters/seg.	32	10	32
max.sectors/seg.	192	240	96
max.sectors/file (including RIB's)	9600	30,240	800
directory search	HDI	HDI	DMB

	DOS.D	DOS.E	DOS.G
min.processor req.	48K 5500	48K 5500	60K 1800
disk controller/drive type used	9370, (9374) [9390]	9350 9354	1840
phys. drives on system	2-8 (2-8) [2-3]	2-4	2-4
type of disk	Scotch 911-0 or equivalent (Datapoint model 80428) [Scotch 949/80 or equivalent]	Scotch 92-204 or equivalent	Maxell FD-3200S floppy disk or equivalent
logical drives per physical drive	2 (2) [5]	1	1
cylinders used/drive	253 (203) [253]	203	76
tracks used/drive	2,024 (1624) [2024]	406	76
sectors/track	24	24	26
sectors/drive	48,576 (38976) [48,576]	9,744	1,976
bytes/drive	12,435,456 (9,977,856) [12,435,456]	2,494,464	505,856
user sectors/drive	48,000 (38,400) [48,000]	9,600	1600
user bytes	12,288,000 (9,830,400) [12,288,000]	2,457,600	405,600
sectors/cluster	24	6	3
clusters/track	1	4	8
clusters/cylinder	8	8	8
max.clusters/seg.	32	32	32
max.sectors/seg.	768	192	96
max.sectors/file (including RIB's)	up to 48,000	9600	1600

directory search HDI HDI DMB

In the table below "cylinder" refers to the PDA MSB and "sector" refers to the PDA LSB.

	DOS.A/DOS.E	DOS.B	DOS.C	DOS.D	DOS.G
PDA of CAT	0,0	0,0	041,0303	0,0	041,0343
PDA of CAT Backup	0,0200	0,040	042,0303	0,040	042,0343
PDA of Lockout CAT	0,1	0,1	043,0303	0,1	043,0343
PDA of Lockout CAT Backup	0,0201	0,041	044,0303	0,041	044,0343
PDA of HDI	0,2	0,2	NA	0,2	NA
PDA of HDI Backup	0,0202	0,042	NA	0,042	NA
Directory Location	cylinder 0 sectors 6 to 025	cylinder 0 sectors 5 to 024	sector 303 cylinders 021 to 040	cylinder 0 sectors 7 to 026	sector 343 cylinders 021 to 040
Directory Backup Location	cylinder 0 sectors 046 to 065	cylinder 0 sectors 045 to 064	sector 303 cylinders 01 to 020	cylinder 0 sectors 047 to 066	sector 343 cylinders 01 to 020

APPENDIX G. DISK DATA FORMATS

G.1 Disk Data Formats

The DOS itself does not deal with the user's data below the record level. It only keeps track of where the records are, allowing the user to format the data in any manner he pleases. The user is presented with records that are 253 bytes long. The system keeps the physical file number in the first physical location of each sector and the system logical record number of the given record in the second (LSB) and third (MSB) physical locations of each sector. This is done to insure that the record obtained is the record desired. The last 253 bytes may contain anything the user chooses. There are, however, some assumptions made by the DOS and the programs supplied with it that deal with disk data. These assumptions fall into several classes described below. The two types normally of greatest interest are object records and symbolic data records. Object records include all records that are to be loaded into memory by the DOS loader. Symbolic data records include all records that are to be handled by the standard data handling programs. These programs include the general purpose editor, the assembler, DATASHARE, RPG II, DOSBASIC, and the DATABUS programs (both source lines for the various compilers and data records handled by the resulting programs).

G.2 OBJECT File Format for Disk

Object files contain binary data which can be loaded using the system loader and then executed. Multiple logical records can be grouped into one physical block.

BYTE	CONTENTS
1	0 => object record, or 0377 => end of block
2	H - load address for record
3	L
4	-H - ones complement of load address
5	-L
6	count of data bytes following

End-of-file is indicated when the count byte has a value of zero. For the end-of-file record, the value of HL is the entry

point address of the object code. The object file created by the ASSEMBLER has a system loader object format.

Logical Record Number	Byte #	Description
LRN 0 (RIB)	0	Physical File Number
	1	Logical Record Number (LSB)
	2	Logical Record Number (MSB)
	3	0377
	4	Segment Descriptor 1
	5	
	5	Segment Descriptor 2
	.	
	.	
	.	
	2N+2	Segment Descriptor N
	2N+3	
	2N+4	0377
	2N+5	0377
LRN 1 (RIB COPY)		
LRN 2	0	Physical File Number
	1	Logical Record Number (LSB)
	2	Logical Record Number (MSB)
	3	0 - indicating data block
	4	Starting address of block (LSB)
	5	Starting address of block (MSB)
	6	One's complement of LSB of starting address
	7	One's complement of MSB of starting address
	8	Block length (n)
	9	Beginning of data
	.	
	.	
	.	
	n+9	0 - Next data block
	n+10	Starting address of block (LSB)
	n+11	Starting address of block (MSB)
	n+12	One's complement of LSB of starting address
	n+13	One's complement of MSB of starting address
	n+14	Block length (m)
	n+15	Beginning of block data
	.	
	.	

```
                    .
                    n+m+15- Next data block
                    .
                    .
                    .
                    .       0377 - End of Record

LRN 3               0       Physical File Number
                    1       Logical Record Number (LSB)
                    2       Logical Record Number (MSB)
                    3       0 - Next data block
                    .
                    .
                    .

LRN N               0
                    .
                    .
                    .
                    .       0 - Last data block
                    .       Transfer address (LSB)
                    .       Transfer address (MSB)
                    .       One's complement of the LSB of the
                            transfer address
                    .       One's complement of the MSB of the
                            transfer address
                    .       0 - block length equal to zero signifies
                            end-of-file
```

G.3 Relocatable Code Formats

Relocatable object code is initially assumed to be starting at location 010000 until a "select new PAB" or "select new location" code is encountered.

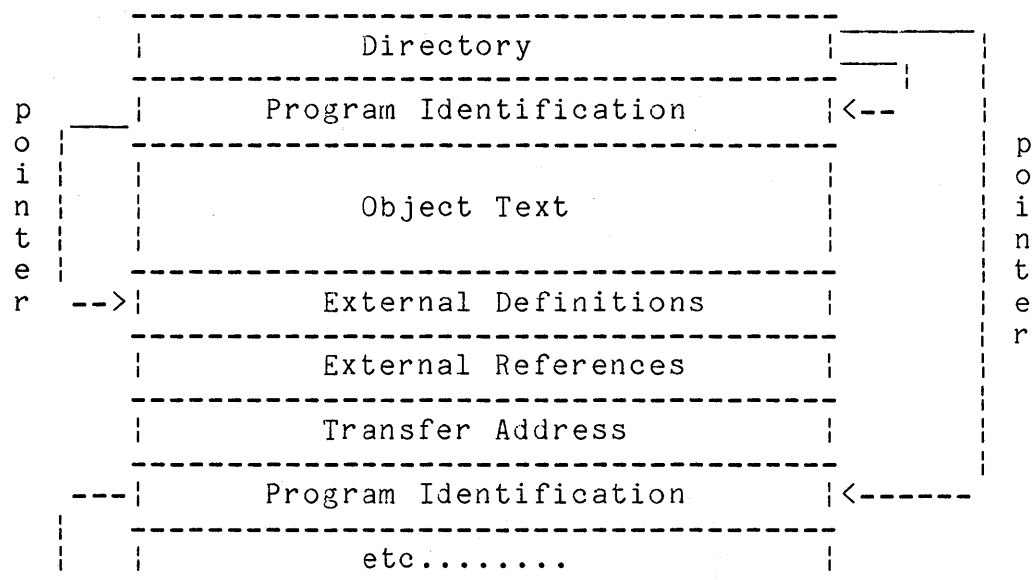
Each sector containing relocatable code starts with a one byte header containing sector contents code. The relocatable code in each sector is followed by a byte containing binary zero.

Sector contents codes are:

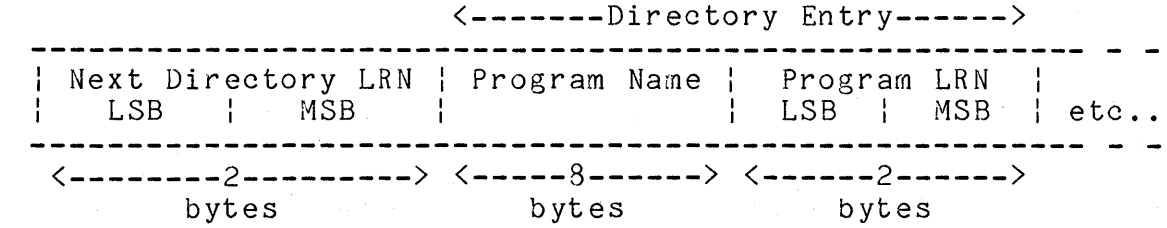
```
0200  Directory
0201  Program Identification
0202  Object Text
0203  External Definitions
0204  External References
```

0205 Transfer Address

Relocatable code files are in library form as follows:

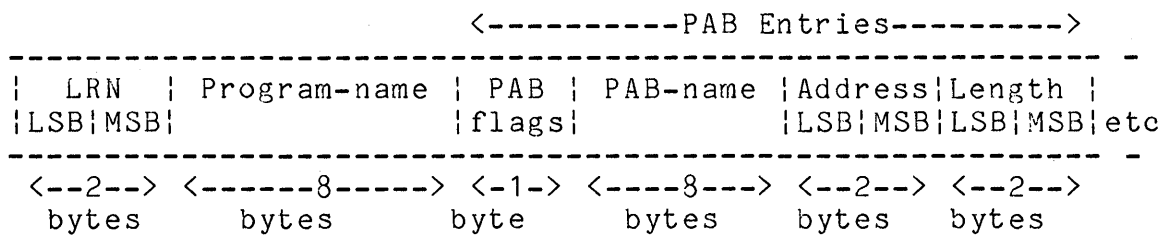


G.3.1 Directory



A directory entry is required for each object program in a library. The first sector of the object code library is reserved as a directory for the first twenty-four programs in the library. If the library contains more than twenty-four programs, a pointer is generated that points to the LRN of the next directory sector (the sector following the twenty-fourth object program). The last directory sector used has a pointer set to 0377, 0377.

G.3.2 Program Identification

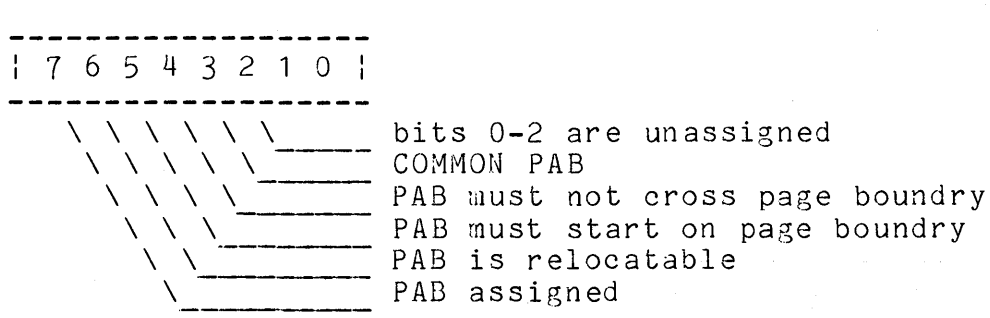


LRN is a pointer to the first sector following object text (the first external definition sector, or the first external reference sector, or the transfer address if there are no definitions or references).

The program name is an eight character name of the program, as reflected in the program id record.

Each PAB (program address block) defines a separate address counter used to assign memory locations. Up to fifteen PAB's can be defined for each program (PAB numbers 1-15). Flag bits are used to indicate relocatability and page sensitivity.

PAB flags:



G.3.3 Object Text

Relocatable object text is interspersed with control bytes used by the linkage editor in creating absolute code.

G.3.3.1 Memory Location

Codes 0160 and 0161 are used to define starting memory locations.

Select New PAB

0160	PAB

PAB defines the number of the Program Address Block to be used for the object code that follows. If the PAB is not in use, the new location will be zero.

Select New Location

0161	LSB	MSB

LSB and MSB define the new location in the current PAB of the next byte of object code.

G.3.3.2 Absolute Text

Codes 0001-0077 precede code and data that does not require relocation.

Absolute Text

1-0077	1-63 absolute text bytes

The code is a count of the number of absolute text bytes that follow.

G.3.3.3 Complex Relocatable References

Codes 0100-0157 are used to define operators and operands of complex expressions that are evaluated by the linkage editor during relocation. Complex expressions are in encoded Polish Postfix notation.

Push Relocatable Location on Stack

0100+PAB	LSB	MSB

PAB, LSB and MSB define the assembled memory location.

Push External Reference on Stack

0120+MSB	LSB

MSB and LSB are an index to an external reference entry.

Push Binary Value on Stack

0140	LSB	MSB

LSB and MSB are a 16 bit binary integer.

Operators:

<	>	.OR.	.XOR.
0141	0142	0143	0144

.AND.	+	-	*
0145	0146	0147	0150

/	Negate	.MOD.
0151	0152	0153

Codes 0141-0153 are expression operators.

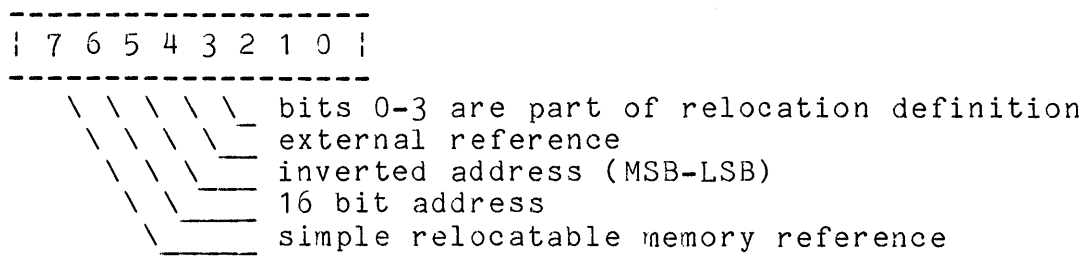
Pop Result of Evaluation from Stack:

Pop LSB	Pop MSB	Pop LSB-MSB	Pop MSB-LSB
0154	0155	0156	0157

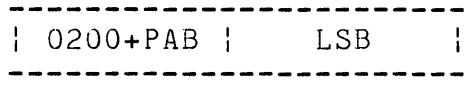
Codes 0154-0157 terminate evaluation of complex expressions and indicate the form of the absolute code to be generated.

G.3.3.4 Simple Relocatable References

Codes 0200-0377 are used for simple relocatable references consisting of a single relocatable symbol or relocatable symbol plus a non-relocatable displacement. Codes for simple relocation can be decoded as follows:



LSB Reference

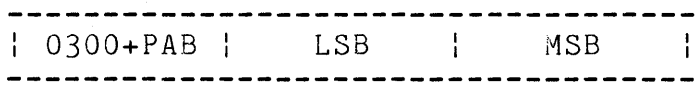


LSB defines the relocatable memory location.

MSB Reference

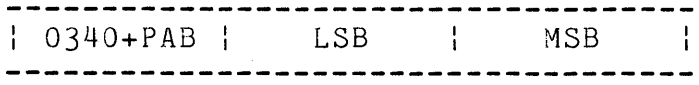
PAB, LSB and MSB define the relocatable memory location. A full sixteen bit address must be given in case a carry occurs between LSB and MSB during relocation.

LSB-MSB Reference



PAB, LSB and MSB define the relocatable memory location.

MSB-LSB Reference



PAB, MSB and LSB define the relocatable memory location.

LSB External Reference

0220+MSB	LSB

MSB and LSB are an index to an external/forward reference entry table.

MSB External Reference

0260+MSB	LSB

MSB and LSB are an index to an external/forward reference entry table.

LSB-MSB External Reference

0320+MSB	LSB

MSB and LSB are an index to an external/forward reference entry table.

MSB-LSB External Reference

0360+MSB	LSB

MSB and LSB are an index to an external/forward reference entry table.

G.3.4 External Definitions

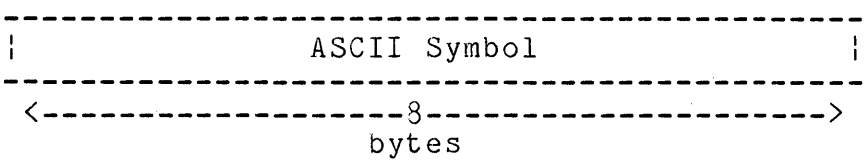
External name	PAB or 0200	LSB	MSB
<-----8----->	<-----1----->	<-1->	<-1->
bytes	byte	byte	byte

External definitions are external symbols made available to other relocatable modules. External references made by other relocatable modules are linked to external definitions as

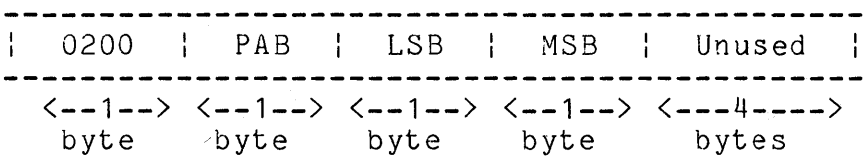
discussed in Chapter 1. The location of each relocatable external definition is defined by PAB, LSB and MSB. A flag (0200), LSB and MSB define non-relocatable external definition values. Up to twenty-two external definitions can be defined in each external definition sector. All external definition sectors for a given program must be contiguous, and not intermixed with external reference sectors.

G.3.5 External and Forward References (4096 maximum)

External Reference



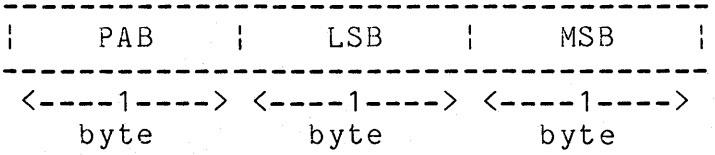
Forward reference



A forward reference is defined as a reference whose value is unknown at some given time in the relocatable module's creation, but whose value is known later, and then is plugged into the forward reference table.

All external reference/forward definition sectors must be contiguous.

G.3.6 Transfer Address



PAB, LSB and MSB define the starting location in the program. If PAB=0377, a starting location was not specified.

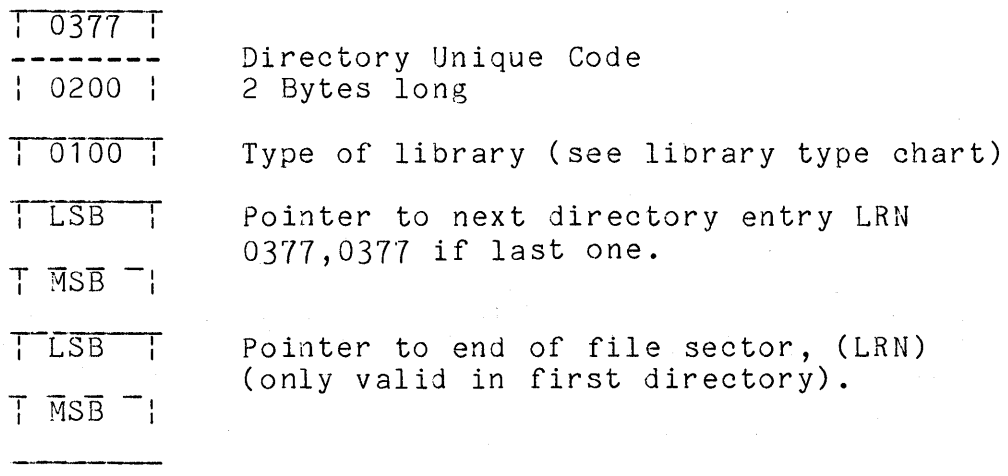
G.4 Format of Library Files

The Library is constructed from two types of entries, Directory Entries and members.

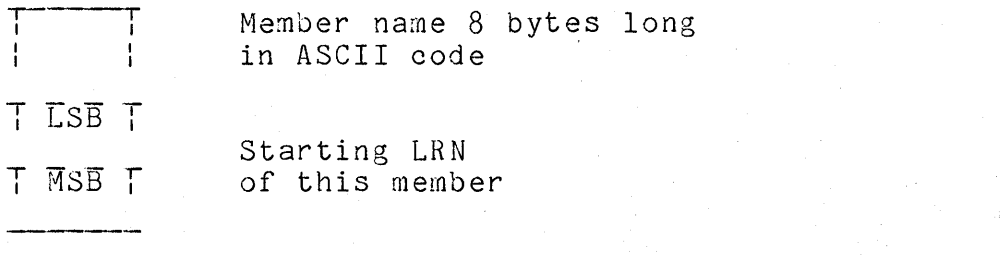
G.4.1 Directory

The first entry of the library file must be the first Directory Entry. Additional directory entries are formatted as required and linked into the directory chain. Each directory has two major parts:

1) The directory header which is 7 bytes. The format is as follows:

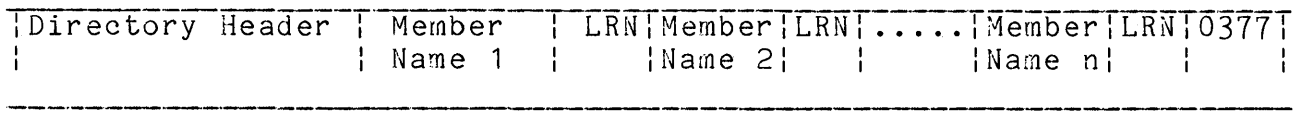


2) Member name entries, each one is 10 bytes long.



One directory entry can contain a maximum of 24 member names. All unused member name entries will be set to 0377's. A deleted member will be set to 0377's.

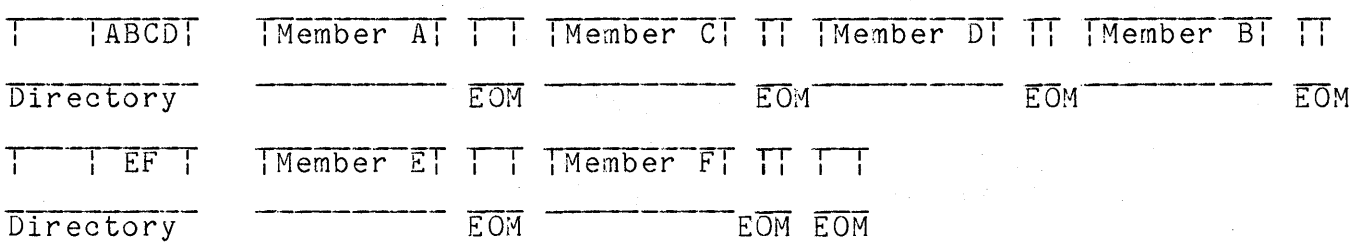
An entire directory entry:



G.4.2 Members

The members are the second type entry of the library. Each member is pointed to by the member name pointer in one of the directory entries. Each member is terminated by an end of member (EOM) code. The EOM is indicated by a sector which contains six bytes of 000 followed by 010.
NOTE: EOM indicates only the end of this member not the end of the library.

A simple library file format



G.4.3 Library Type Chart

If the library contained more than 24 members another directory entry would be placed into the chain of directories.

The following is the bit chart for library types

```
1.  ...   ...   Reserved
.1  ...   ...   Absolute
..  1..   ...   Relocatable
..  .1.   ...   Dataform
..  ..1   ...   Databus (DBL)
..  ...   111   Reseved for future use
```

G.5 DATABUS Code File Format

DATABUS files contain code produced by the DATABUS compiler for use by its interpreter. All blocks are 251 bytes long.

BYTE	CONTENTS
1	040 - DATABUS code file indicator
2	H - load address
3	L
4	-H - complement of load address
5	-L

End of file is indicated by bytes 1 through 6 being binary zeros, followed by a binary three.

G.6 DATAFORM Data File Format

Every record created by a DATAFORM form is stored consecutively on the disk terminated with a 015 designated as the end of logical record character. Disk sector boundaries are transversed by placing a 003 to represent the end of physical record. An end of file mark is six zeros followed by a 003 beginning at the start of the next unused sector. This complies with Datapoint's DOS text file structure. However, other characters immediately following the 003 are necessary record descriptors to allow record form linkages and the record backspace feature to be implemented in DATAFORM. The first character following the end of physical record character, 003, represents the form number that created the first logical record starting in

that sector, biased by 4. The character immediately following is the absolute address in the sector of the first character of the logical record. (Note that the first data character of every sector starts at address 003.) There must be a similar pair of characters describing every logical record that starts in that sector. These character pairs must be in that sector and in consecutive order. (i.e. The first pair relates to the first record, the second pair to the second record, etc.) The remainder of the sector, if unused, is filled with zeros. DATAFORM 1100 Version 1 may use the entire 253 bytes available in the sector. However, DATAFORM Version 2 does not use the last two bytes of every sector, only 251 bytes are used.

G.7 MULTIFORM File Format

The first sector of a Multiform file contains information concerning the file name, form library relating to the data, and end of file position. The format of this header sector is described below. The first byte of a sector has a sector address of zero.

SECTOR ADDRESS	DESCRIPTION
0- 2	Reserved for DOS
3	Contains a byte value of 003
4- 11	First 8 characters of the data file name
12- 14	Three characters of the data file extension
15- 16	LRN of the last sector which has data written to it. Must be in LSB,MSB format
17- 24	First 8 characters of the form file name
25- 27	Three characters of the form file extension
28-251	Not care conditions
252	Contains a byte value of 000
253	Contains a byte value of 003
254-255	Reserved for DOS

All records are now written consecutively in a non-space compressed format. Each record is terminated by an 015. The end of the physical record is indicated by an 003. Bytes after the 003 contain special information that Multiform uses. This information is right justified in the sector, which will be described from right to left.

SECTOR ADDRESS	DESCRIPTION
253	Contains a byte value equal to the number of records that start in that sector plus the value 3. Each record that starts in the sector has two bytes that describe its position and the form that created it.
252	Contains a byte value equal to the form number of the form that relates to the last record that starts in the sector.
251	The true sector position of the last record that starts in this sector.

The next preceeding byte pair describes the next to the last record that starts in that sector in the same format as described above. These byte pairs are repeated for every record that starts in that sector. The end of physical record preceeds these record description byte pairs by no more than one character. The exception to this is the last sector in the file which contains data. In this case, immediately preceeding the record description byte pairs will be a byte whose value is the true sector address of the end of physical record character. Note, if no record begins in this sector, sector byte address 253 will contain an 003 and the preceeding byte will have the sector address of the end of physical record character. The next sector in the file will contain the standard DOS end-of-file mark.

G.8 TEXT File Format

TEXT files typically contain data, source statements, or whatever is meaningful to the user. The requirement is that the data contained in the text file must be equal to or greater than 040 (space). The only bytes less than 040 which are allowed are the following:

CONTROL BYTE	SYMBOL	MEANING
000		NULL. The NULL control byte is used in the indication of the end of the file.
003		END-OF-MEDIUM. No more meaningful data is contained

in this block. The EM is NOT a data byte but must be within the block.

011<cnt> SPACE COMPRESSION. The byte following the 011 is a binary count of spaces which have been compressed. <cnt> can be between 2 and 255, inclusive. The 011<cnt> sequence must not be split across block (sector) boundaries.

015 END-OF-RECORD. The EOR, also the Enter [ENT] or Carriage Return character, indicates the end of the logical record. It is NOT a data byte.

032 DELETE. The DEL byte indicates the data byte is deleted. The DEL is NOT a data byte. Entire records (including the EOR indicator) can be deleted by over-writing them with DEL bytes.

There is no explicit maximum size for a logical record. A logical record can span as many blocks (sectors) as necessary, within the capacity of the device. A physical block must be less than or equal to 251 bytes, including any necessary EOR bytes and the trailing EM byte. Text files can be either space or record compressed, or both, or may be blocked. Space compressed files use the CMP control bytes to represent strings of blanks within logical records. A space compressed file has no particular relationship between the physical and logical records. A record compressed file does not use space compression, but uses EOR control bytes to identify logical records within physical records. A record compressed file has no particular relationship between the physical and logical records. A blocked file has some fixed relationship between logical and physical records, normally containing one logical record per physical sector. Datapoint text files are generally both space and record compressed, record compressed only, or blocked. Blocked files with space compression are possible but are not generally used.

End of file is indicated by bytes 1 through 6 being binary zeros <NUL>, followed by a binary three <EM>.

G.9 ISI File Format

The indexed file is a normal GEDIT-compatible text file. The ISAM file is of the following format:

First record - header record

```
0-10 - indexed file name of form filenameext
  11 - PFN of the ISAM file
  12 - the sector of the ISAM file RIB
  13 - the cylinder of the ISAM file RIB
  14 - PFN of the indexed file
  15 - the sector of the indexed file RIB
  16 - the cylinder of the indexed file RIB
  17-18 - OBSOLETE
  19 - OBSOLETE
  20-22 - last record used in data file (BUFADR, LRN LSB, LRN MSB)
  23-25 - next free entry in ISAM file (BUFADR, LRN LSB, LRN MSB)
```

Second sector - highest level

The highest level index is a single sector using the same format as the intermediate level sectors described below.

Third+ sectors - lowest level

```
KEY/015/NEXBUF/NEXSEC/NEXCYL/RECBUF/RECLSB/RECMSB//KEY....
Since key cannot be split over sector boundary,
sector is padded with 0377's.

KEY - uncompressed ASCII key with trailing spaces truncated
      0 -> first record
      0377 -> last record
NEXBUF - buffer address of the next key, 0 implies next sequential
NEXSEC - sector address of the next key
NEXCYL - cylinder address of the next key
RECBUF - buffer address of the indexed record
RECLSB - logical record number LSB of the indexed record
RECMSB - logical record number MSB of the indexed record
```

N+ sectors - intermediate levels

```
KEY/012/NEXSEC/NEXCYL//KEY...
 Since key cannot be split over sector boundary, sector is filled
 0377's.

KEY - uncompressed ASCII key with trailing spaces truncated
      0 -> first record
```

0377 -> last record
NEXSEC - sector address of the next-lower-level key
NEXCYL - cylinder address of the next-lower-level key

The ASCII keytag file produced by SORT or INDEX utilities and used as an intermediate step in the creation or re-creation of an ISI file has the following format.

RECLRN/RECBUF/KEY/015//KEY...

RECLRN - 5 byte ASCII decimal logical record number of the indexed key
RECBUF - 3 byte ASCII decimal buffer address of the key
the ASCII decimal numbers have leading blanks
KEY - compressed ASCII key with trailing spaces truncated

G.10 SORT TAG File Format

The format of a SORT TAG file is as follows.

1. For each record in the corresponding input data file, the TAG file will have a three byte binary pointer to the first byte of the record.

2. The format of the pointer is:
Byte 1: MSPLRN (Most significant portion of LRN),
Byte 2: LSPLRN (Least significant portion of LRN),
Byte 3: BUFTPTR (Disk buffer pointer).

3. The three-byte binary pointers are blocked 83 to a physical disk record.

4. The physical-end-of-record is indicated by an <EM>.

5. The end-of-file is indicated by bytes 1 through 6 being <NUL>s followed by one <EM>.

Manual Name_____

Manual Number_____

READER'S COMMENTS

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____

All comments and suggestions become the property of Datapoint.

CUT ALONG LINE FOR MAILING

Fold Here

Fold Here and Staple

FIRST CLASS
Permit
5774
San Antonio
Texas

BUSINESS REPLY MAIL
No Postage Necessary if mailed in the United States

Postage will be paid by:

DATAPOINT CORPORATION
DIRECTOR OF SOFTWARE SUPPORT
8550 Datapoint Drive, Mail Station# N60
San Antonio, Texas 78284